

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Проектирование автоматизированных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту  
на тему

**АВТОМАТИЗИРОВАННАЯ СИСТЕМА УПРАВЛЕНИЯ ПОЛОЧНЫМ  
ПРОСТРАНСТВОМ СУПЕРМАРКЕТА**

БГУИР КП 1-53 01 02 06 014 ПЗ

Студент

Руководитель

В. Г. Колесников

А. Ф. Трофимович

Минск 2019

## СОДЕРЖАНИЕ

Введение.....	6
1 Системно-аналитическая часть.....	7
1.1 Описание и анализ объекта автоматизации.....	7
1.2 Постановка задачи.....	10
2 Проектирование системы .....	14
2.1 Концептуальная модель системы .....	14
3 Реализационная часть .....	30
3.1 Выбор инструментальной платформы и комплекса технических средств.....	30
3.2 Программное обеспечение .....	39
3.3 Организационное обеспечение .....	53
Заключение .....	60
Список использованных источников .....	61
Приложение А (справочное) Листинг кода.....	62

## ВВЕДЕНИЕ

Довольно безбедное существование (в материальном плане) современному человеку обеспечила научно-техническая революция, которая началась в середине прошлого столетия и привела к коренному изменению технических средств производства на основе механизации и автоматизации.

Автоматизация производства – это широкое использование в производственных процессах автоматического и автоматизированного оборудования, в котором функции управления и контроля переданы управляющим и автоматическим устройствам (автоматам).

Слово «автомат» в переводе с греческого означает «самодействующий». В Древней Греции так назывались механизмы и устройства, способные самостоятельно, без видимого участия человека выполнять некоторые простые действия [1].

Следует отличать автоматизированное управление от автоматического.

Автоматическое управление – процесс управления происходит без вмешательства человека – автоматически.

Автоматизированное управление – человек участвует в процессе управления как элемент системы управления (выбор управляющего воздействия, принятие решения об указании управляющего воздействия) [2].

Автоматизация розничной торговли — это комплекс мероприятий, включающий внедрение специализированного оборудования и программных продуктов для значительного повышения эффективности деятельности торгового предприятия. Наиболее популярными проблемами в розничных магазинах являются медленное и некачественное обслуживание покупателей, отсутствие достоверного учета товародвижения, неграмотное управление ассортиментом и ценообразованием, оседание товара на складе, злоупотребления и ошибки персонала, хищения товара. Это далеко не весь список рисков, с которыми приходится сталкиваться магазину. И самое верное решение в данной ситуации — комплексная автоматизация торговли [3].

В рамках курсовой работы было принято решение разработать автоматизированную систему управления полочным пространством супермаркета. Проблема является актуальной и требует комплексного подхода ко всем ее частям. Благодаря ее углубленному анализу и разработке отдельных компонентов можно будет выяснить наиболее подходящий вариант реализации.

# 1 СИСТЕМНО-АНАЛИТИЧЕСКАЯ ЧАСТЬ

## 1.1 Описание и анализ объекта автоматизации

Для любого розничного торговца “полка” - это самое дорогое, что у него есть. Именно поэтому, он стремится разместить на имеющейся площади как можно больше торгового оборудования и получить с каждого сантиметра торговой площади как можно больший доход. Для того, чтобы эффективно управлять внутренним пространством магазина, необходимо учитывать огромное количество факторов [4].

Один из немаловажных моментов в управлении полочным пространством – постоянное заполнение полочного пространства. Так как посетители магазина берут товар, который им нужен и который им ближе, нередко возникает необходимость в “приближении” этого товара к покупателю. Кроме того, важно своевременно заполнять полку, на которой закончился товар определенного вида или подавать сигнал управляющему персоналу о том, что товара нет как на полке, так и на складе, для своевременного принятия решения.

Чтобы облегчить работу по постоянному контролю за заполнением полочного пространства возможно внедрение автоматизированной системы. Данное решение предоставляет возможность разгрузить персонал от постоянного контроля полок, а также увеличить скорость появления товара при его отсутствии, когда имеется возможность быстро его восполнить.

Существуют системы с похожей логикой выполнения процедур. Кроме того, они дают возможность сразу увидеть процесс заполнения и предоставляют интуитивно понятный интерфейс для конфигурации системы.

Наиболее популярной в данной области является украинская компания *ABM Cloud*, которая предоставляет систему управления полочным пространством или по-другому – систему мерчендайзинга торгового зала *ABM Shelf*.

Цель системы *ABM Shelf*— повышение продаж и прибыли, оптимизация расходов на хранение товара, высвобождение замороженных средств. Цель достигается путем эффективного управления торговым пространством розничной сети и автоматизации управления выкладкой товаров. Автоматизация управления выкладкой и торговым залом с *ABM Shelf* позволяет моделировать торговый зал магазина в 3D, анализировать продажи, контролировать выкладку.

### Основные функции системы *ABM Shelf*:

- Конструктор оборудования с возможностями *3D* моделирования различных видов торгового оборудования с учетом габаритных характеристик (длина, глубина, высота и др.) и создания произвольного количества шаблонов для торгового оборудования с фиксированными размерами;
- Эффективное управление торговым пространством;
- Управление торговым залом;
- Работа с базой данных магазинов (готовый перечень), удобный поиск и навигация;
- Создание моделей торгового зала для магазинов (неограниченное количество);
- Аналитика продаж торгового зала (*ABC* анализ, анализ продаж);
- Автоматизация управления выкладкой и управление торговым оборудованием;
- Разработка планограммы выкладки товара;
- Работа с несколькими стеллажами одновременно;
- Удобная визуальная аналитика по данным продаж (*ABC*, суммы продаж) и выкладки (по производителям, по полкам, по габаритам, стоимости, остаткам) с возможностью экспорта в Excel [5]. Сайт системы представлен на рисунке 1.1.

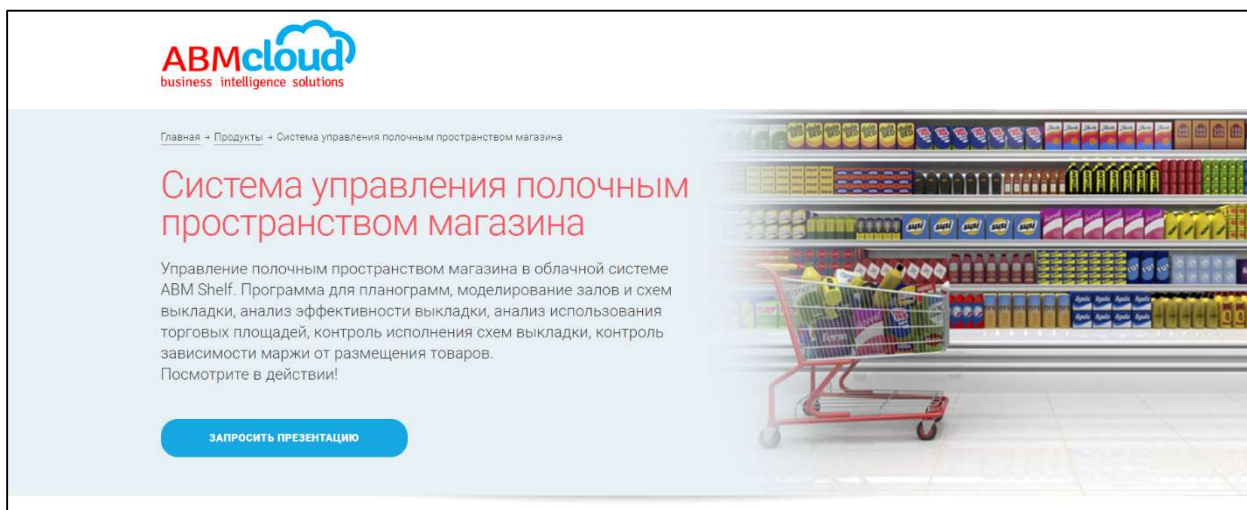


Рисунок 1.1 – Сайт системы *ABM Shelf*

Российская компания, предоставляющая похожую автоматизированную систему – АСТОР с продуктом *RS.ShelfSpace*.

*Retail Suite Shelfspace* — это система управления полочным пространством магазина. Она позволяет автоматизировать и настроить эффективное управление торговым пространством розничной сети. Реализован сценарный подход, который позволяет одновременно управлять несколькими задачами, запущенными параллельно. Торговое пространство розничной сети должно быть эффективным и управляемым. Для того, чтобы этого добиться, *RS.ShelfSpace* позволяет контролировать целый набор показателей визуального мерчандайзинга, отражающих качество использования торговых площадей.

При моделировании розничного пространства, *RS.ShelfSpace* обрабатывает сотни и тысячи объектов. Управление полочным пространством магазина выполняется централизованно. Выкладка товаров осуществляется по рассчитанным системой планограммам, построение которых, в свою очередь, ведется по настроенным правилам в автоматическом режиме.

Реализована всеобъемлющая поддержка как регулярных продаж, так и промо-акций. Простая и удобная работа с планограммами и реалограммами, хранение истории изменений.

Поддержка открытия новых магазинов. План торгового зала создаётся в рамках бизнес-процесса открытия нового магазина, и в дальнейшем этот план поддерживается в актуальном состоянии с учётом всех изменений торгового пространства и торгового оборудования.

И, конечно, система управления мерчандайзингом предоставляет разнообразную аналитическую отчётность, позволяющую оценить эффективность торговых мероприятий.

Основные преимущества системы *RS.ShelfSpace*:

- сокращение затрат на персонал центрального офиса и магазинов;
- сокращение капитальных затрат;
- повышение лояльности покупателя, увеличение продаж;
- улучшение контроля работы магазина.

Система управления *RS.ShelfSpace* позволяет контролировать набор показателей, отражающих эффективность использования торгового пространства. Каждый из показателей может быть детализирован до конкретной секции или оборудования [6]. Сайт системы представлен на рисунке 1.2.

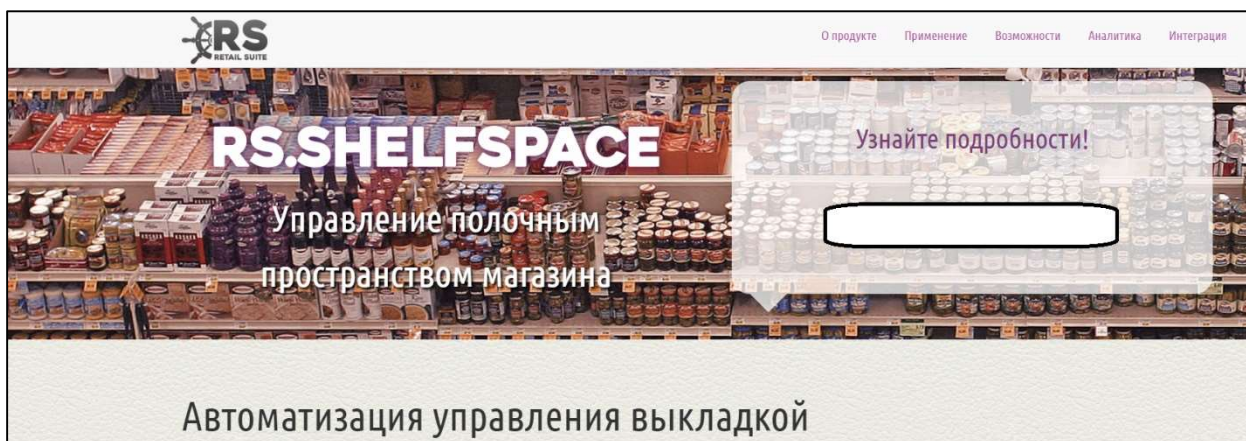


Рисунок 1.2 – Сайт системы *RS.ShelfSpace*

Данные системы обладают рядом преимуществ и недостатков. Следует учесть их при разработке курсового проекта, так как некоторые из возможностей систем не соответствуют идее реализации данной работы.

Оба продукта предоставляют возможность управления полочным пространством. Они обеспечивают визуальное размещение товара на определенных позициях, а так же упрощают процесс управления выкладкой, предоставляя готовые решения расположения. Кроме того, они выполняют все это в централизованном режиме с возможностью одновременной параллельной работы с несколькими позициями.

Данные продукты являются облачными решениями. Взаимодействия с оборудованием как такового нет, то есть выкладкой товаров по прежнему занимается персонал, несмотря на разнообразную аналитику и выработку решений. Также у данных продуктов нет связи со складскими запасами продуктов, что может повлиять на малую заполненность полок в связи с отсутствием определенных видов товаров на полках и несвоевременным заполнением.

## 1.2 Постановка задачи

Супермаркет нуждается в автоматизации полочного пространства с целью сократить расходы на персонал и улучшить качество предоставляемых услуг. Имеется большой стеллаж, к которому возможно подключение системы без видимых для клиентов компонентов. Кроме того, директор данного супермаркета обладает складом для хранения товаров с возможностью внедрения системы автоматической доставки к стеллажам.

Система должна обладать возможностью визуализации происходящего процесса, а также оповещать управляющий персонал о каких-либо сбоях и проблемах. В случае наличия возможного варианта восстановления работоспособности, система должна применять его, при этом сообщив о случившемся в отчете о своей работе.

Необходимо, чтобы система обладала возможностью разграничения прав доступа к отдельным своим частям: менеджеры и системный администратор должны иметь доступ ко всем ее частям, авторизованные пользователи – ко всем страницам, кроме страницы с информацией о грузоперевозках, другие пользователи – только к домашней странице, странице с контактными данными и странице, на которой кратко описывается система.

Система должна просчитывать и выдавать в отчет некоторые аналитические данные в процессе работы. В случае простоя, она должна переходить на экономный режим работы, в случае большой загруженности – увеличивать потребляемые мощности. Необходимо обеспечить минимальный набор данных, которые будут описывать систему, а также предоставлять контактные данные владельца магазина.

Также система должна иметь грамотно спроектированную базу данных для своевременного и качественного выполнения возложенных на нее задач хранения и предоставления данных.

### **1.2.1 Требования к аппаратной части реализации системы**

**Процессор CPU.** Требуется x86- или x64-разрядный двухъядерный процессор с тактовой частотой 2.2GHz или выше, с набором инструкций SSE2.

**Оперативная память.** Требуется ОЗУ объемом не менее 2 ГБ.

**Дисплей.** Требуется VGA монитор с разрешением 1024 x 768.

**Требования к сети.** У сети должна быть пропускная способность более 50 КБ/с (400 КБ/с) и задержка менее 150 мс.

При запуске разрабатываемого приложения на компьютере, не соответствующем рекомендуемым требованиям, производительность работы может оказаться недостаточной. Дополнительно удовлетворительную производительность можно получить используя системы, которые имеют аппаратную конфигурацию, отличную от систем, опубликованных здесь. Например, система с современным четырехъядерным процессором, более низкой тактовой частоты и большей ОЗУ. Кроме того, для взаимодействия с



системой могут быть использованы современные мобильные устройства на базе Android или iOS.

### 1.2.2 Требования к программной части реализации системы

**Операционная система.** Требуется предустановленная ОС Windows 10, Windows 8.1, Windows 8 или Windows 7, для мобильных устройств – Android версии 8 или выше или iOS 8 или выше.

**Клиент.** На персональном компьютере требуется иметь браузер для просмотра сайта. Требование к браузеру: браузер Internet Explorer 10 и выше, другие браузеры последних либо предпоследних версий.

**Сервер приложений.** Сервер реализуется вместе с back-end кодом. Для этого следует использовать Python не ниже версии 3.8 с фреймворком Django не ниже версии 2.2.7.

**Веб-сервер.** Для обслуживания веб-страниц рекомендуется использовать Nginx Server не ниже версии 1.17.5 либо Apache HTTP Server не ниже версии 2.4.41.

**СУБД.** Реализовать систему требуется на базе SQLite не ниже версии 3.30.1.

### 1.2.3 Описание дополнительных спецификаций к проекту

**Функциональные возможности.** Система должна обеспечивать многопользовательский режим работы. Она должна иметь возможность работать одновременно с несколькими клиентами. Однако планируется одновременный доступ малого количества пользователей, поэтому продвинутого оборудования для этих целей не требуется. Так же у пользователей системы должен быть постоянный доступ к данным о загруженности полок или сбоех системы, чтобы вовремя предпринять необходимые шаги для устранения неполадок.

**Удобство использования.** На персональном компьютере требуется иметь редактор кода и браузер для просмотра сайта. На мобильном устройстве требуется только браузер. Требование к браузеру: браузер Internet Explorer 10 и выше, все остальные браузеры последних либо предпоследних версий.

**Надежность.** Система должна быть в работоспособном состоянии во время взаимодействия с полками супермаркета, то есть на протяжении рабочего времени персонала. В остальных случаях для устранения лишних затрат на электроэнергию рекомендуется переводить систему в режим

пониженного энергопотребления. Время простоя во время рабочего дня – не более 5 %. Также необходимо наличие резервного копирования для базы данных.

**Производительность.** Система должна поддерживать одновременную работу всех сотрудников компании, а также такого количества пользователей, которые могут одновременно пребывать в супермаркете и взаимодействовать с рабочей станцией.

**Безопасность.** Система не должна позволять клиентам видеть какие-либо данные о работе системы, изменять ее свойства или вмешиваться в нормальный цикл взаимодействия с каким-либо из ее компонентов. Для этого необходимо скрыть все ее видимые компоненты, а также разграничить доступ на сайт для неавторизованных пользователей. Доступ к изменению свойств системы должен иметь только системный администратор и менеджеры. Доступ к данным системы должен иметь только персонал, а в отдельных случаях – покупатель при разрешении менеджера на это. Доступ к основному функционалу системы должен иметь только авторизованный пользователь.

**Проектные ограничения.** Система должна быть интегрирована с существующей системой супермаркета, не нарушать нормального ее функционирования и поддерживать постоянную связь со всеми компонентами.

## 2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

### 2.1 Концептуальная модель системы

#### 2.1.1 Составление глоссария проекта

Для описания терминологии предметной области составим глоссарий проекта. Он может быть использован как неформальный словарь данных системы. Глоссарий приведен в таблице 2.1.

Таблица 2.1 – Глоссарий проекта.

Бизнес-менеджер	человек, заведующий делами супермаркета, официально представляющий его в деловых отношениях, управляющий остальным персоналом.
Менеджер супермаркета	человек, управляющий ежедневными делами супермаркета, разрешающий споры и конфликты сторон, управляющий персоналом, за исключением бизнес-менеджера, который делегирует свои полномочия в процессе работы.
Системный администратор	человек, ответственный за бесперебойную работу системы, а в случае возникновения неполадок – устраняющий их.
База данных	объект, содержащий всю необходимую информацию для полноценной работы системы.
Мерчендайзер	сотрудник супермаркета, обеспечивающий контроль за правильной и своевременной выкладкой товаров на стеллажах.
Отчет	текстовая информация, содержащая аналитические данные и данные работы системы.
Пользователь системы	любой сотрудник супермаркета, использующий систему.
Постороннее лицо	объект или субъект, непосредственно не относящийся к работе системы, но взаимодействующий с ней.

#### 2.1.2 Проектирование бизнес-логики

Бизнес-логика - в разработке информационных систем - совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика - это реализация правил и ограничений

автоматизируемых операций. Является синонимом термина «логика предметной области» (англ. domain logic). Бизнес-логика задает правила, которым подчиняются данные предметной области [11].

Проще говоря, бизнес-логика - это реализация предметной области в информационной системе. К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

Например, бизнес-логика рассматривает:

- действующих лиц системы (пользователей с разными правами, разные типы существ в мире игры или иные сущности, о которых может говорить обычный человек (не программист), когда формулирует задачу);
- связи между сущностями (опять же понятные обычному человеку, когда он говорит о системе).

Именно исходя из бизнес-логики начинают формировать архитектуру проекта. Созданная модель диаграммы вариантов использования для проекта, представлена на рисунке 2.1.

Действующие лица:

Обычный пользователь – может использовать систему и супермаркет в обычном режиме с минимальным уровнем доступа;

Менеджер – управляет сотрудниками;

Бизнес-менеджер – выполняет роль менеджера, а также управляет делами магазина на внешнем его пространстве, такими как заключение контрактов, найм и увольнение сотрудников и другое;

Заведующий магазином – выполняет роль менеджера, а также управляет внутренними делами магазина, разрешает конфликты между покупателями и магазином, управляет сотрудниками;

Системный администратор – смотрит за общим состоянием системы и устраняет неполадки, возникающие внутри нее;

Мерчендайзер – смотрит за внешним состоянием системы, оповещает о неполадках, помогает системе правильно работать, если имеет возможность делать это;

Постороннее лицо – не имеет прямого отношения к системе, но может взаимодействовать с ней косвенно, т.е. не вмешиваясь в нормальную работу системы;

Водитель – доставляет товары на склад и выполняет грузоперевозки;

Покупатель – постоянно тестирует систему на работоспособность, не зная об этом;

База данных – хранит и предоставляет данные о работе системы и персонала;

Хранилище – хранит товары и предоставляет их по требованию системы, а также запрашивает товар, если его оказывается недостаточное количество и оповещает о неполадках в своей работе;

Магазин – хранит товары на полках и запрашивает их в случае недостаточного количества, а также оповещает о неполадках в своей работе.

Выделяются следующие варианты использования:

Управлять бизнесом;

Управлять магазином;

Управлять персоналом;

Выполнить заказ на грузоперевозку;

Пополнить склад;

Купить что-нибудь для теста системы;

Показать данные;

Показать все страницы;

Показать страницы Home, About и Contact;

Войти в систему;

Устранить неполадки;

Использовать систему;

Быть ответственным за оборудование;

Следить за сроком годности;

Следить за корректной работой системы;

Управлять данными;

Сгенерировать отчет о работе;

Выдать продукт;

Отслеживать состояние полок;

Запросить продукт;

Сообщить о неполадке;

Выдать сигнал о необходимости пополнения.

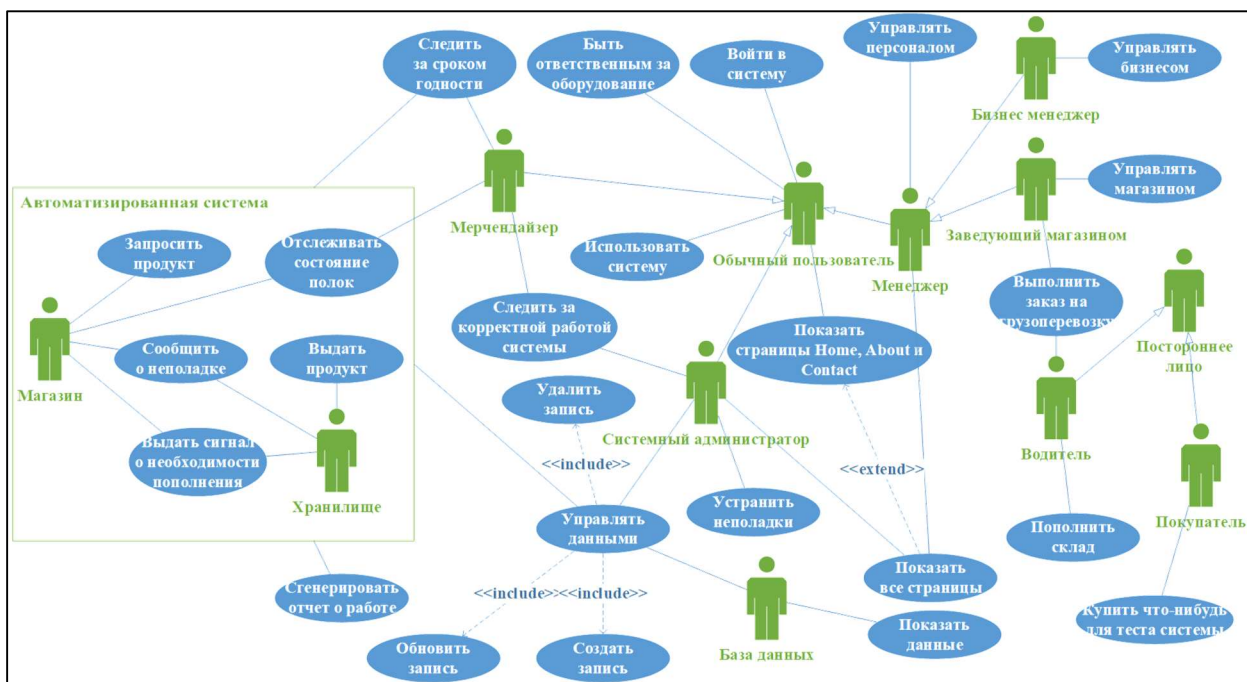


Рисунок 2.1 – Диаграмма вариантов использования

На диаграмме вариантов использования описаны основные требования к системе. Начало действий идет от магазина. В случае недостаточного количества товаров на полке, магазин может запросить ее пополнение. Кроме того он следит за состоянием полок и сообщает о неполадках в случае их возникновения. Хранилище в ответ на запрос о пополнении выдает данный продукт или, в случае его отсутствия, выдает сигнал о необходимости пополнения склада. В свою очередь, персонал супермаркета следит за правильностью работы системы и устраняет неполадки в случае необходимости.

Пользователи могут авторизовываться на сайте системы и просматривать ее текущее состояние. База данных хранит данные о системе и ее работе, выдает их, а также изменяет по запросу от системного администратора или системы. Системный администратор может управлять данными базы данных, а также просматривать все страницы сайта системы, чтобы проверять корректность работы и устранять неполадки в случае их возникновения. Кроме того, он может регистрировать новых пользователей в системе.

Менеджеры управляют персоналом, бизнес-менеджер – делами магазина, заведующий магазином – делами внутри магазина. Водитель выполняет заказы на грузоперевозку и пополняет склад. Покупатель тестирует систему, покупая в магазине необходимый ему товар.

Посторонние лица не должны влиять на систему, поэтому они ничего не делают.

Далее система описывается с помощью нотации IDEF0. В стандарте IDEF0 посредством входа показываются объекты - информационные и материальные потоки, которые преобразуются в бизнес- процессе. С помощью управления показываются объекты - материальные и информационные потоки, которые не преобразуются в процессе, но нужны для его выполнения.

На рисунке 2.2 изображена контекстная диаграмма, представляющая собой самое общее описание системы и ее взаимодействие с внешней средой.



Рисунок 2.2 – Контекстная диаграмма

Для более подробного описания функционирования системы построим диаграмму декомпозиции. Диаграммы декомпозиции предназначены для детализации функций и получаются при разбиении контекстной диаграммы на крупные подсистемы (функциональная декомпозиция) и описывающие каждый подсистему и их взаимодействие. Данная диаграмма представлена на рисунке 2.3

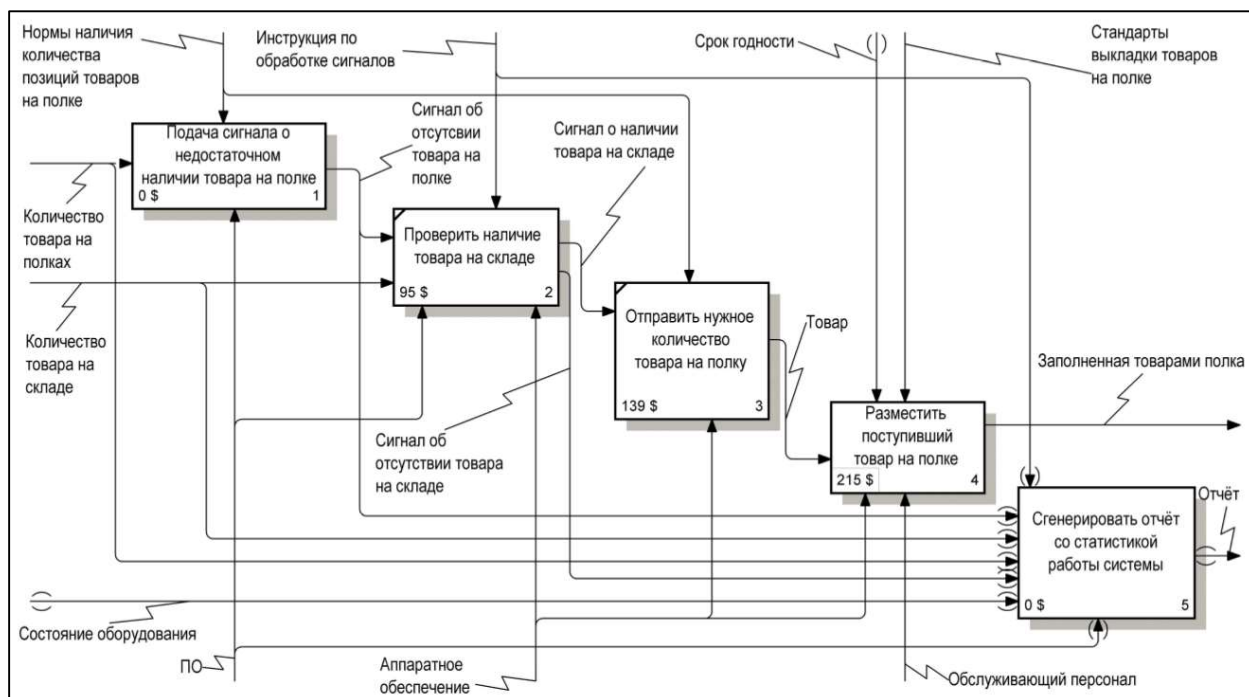


Рисунок 2.3 – Диаграмма декомпозиции системы

На схеме наглядно видно, на каком этапе какие управляющие элементы и какие механизмы задействованы. На вход поступает информация о количестве товара на полках и складе. Когда на полке становится недостаточное количество товаров, подается сигнал об отсутствии товара. Далее проверяется наличие товара на складе. Если товара на складе нет – отправляется сигнал об отсутствии товара на складе. Если товар есть – отправляется сигнал о наличии товара на складе. Нужное количество товара отправляется в магазин, после этого товар размещается на полке. Все важные действия и сбои системы записываются, и в итоге генерируется отчет со статистикой работы системы.

Блок «Подать сигнала о недостаточном наличии товара на полке»: когда на полке магазина не обнаруживается достаточного количества товара для нормальной работы системы в соответствии с нормами наличия количества позиций товаров на полке, выполняется подача сигнала об отсутствии товара на полке. Информация о количестве товара на полках отправляется генератору отчетов.

Блок «Проверить наличие товара на складе»: при поступлении сигнала об отсутствии товара на полке и обработки его в соответствии с инструкцией проверяется наличие товара данной категории на складе. При наличии товара на складе поступает положительный сигнал. Этот и предыдущий блок обрабатывается программным обеспечением. Кроме того, этот блок



задействует аппаратную составляющую системы. Информация о количестве товара на складе и сигнал об отсутствии товара на полке отправляются генератору отчетов.

Блок «Отправить нужное количество товара на полку»: принимает сигнал о наличии товара на складе и обрабатывает его в соответствии с нормами наличия количества позиций товаров на полке. При отправке товара используется аппаратная составляющая системы.

Блок «Разместить поступивший товар на полке»: принимается товар, проверяется его срок годности, и в соответствии со стандартами выкладки товаров при помощи аппаратного обеспечения и, в крайних случаях, при помощи персонала выкладывается на полке. В результате получается заполненная товарами полка.

Блок «Сгенерировать отчет»: принимает данные о количестве товаров на полках и на складе, а также об отсутствии товаров на полках и состоянии оборудования, и генерирует отчет в соответствии с инструкциями и с использованием программного обеспечения.

Далее строится диаграмма декомпозиции для работы «Разместить поступивший товар на полке». Пять блоков на диаграмме обозначают пять этапов выполнения задачи. Разработанная диаграмма изображена на рисунке 2.4.

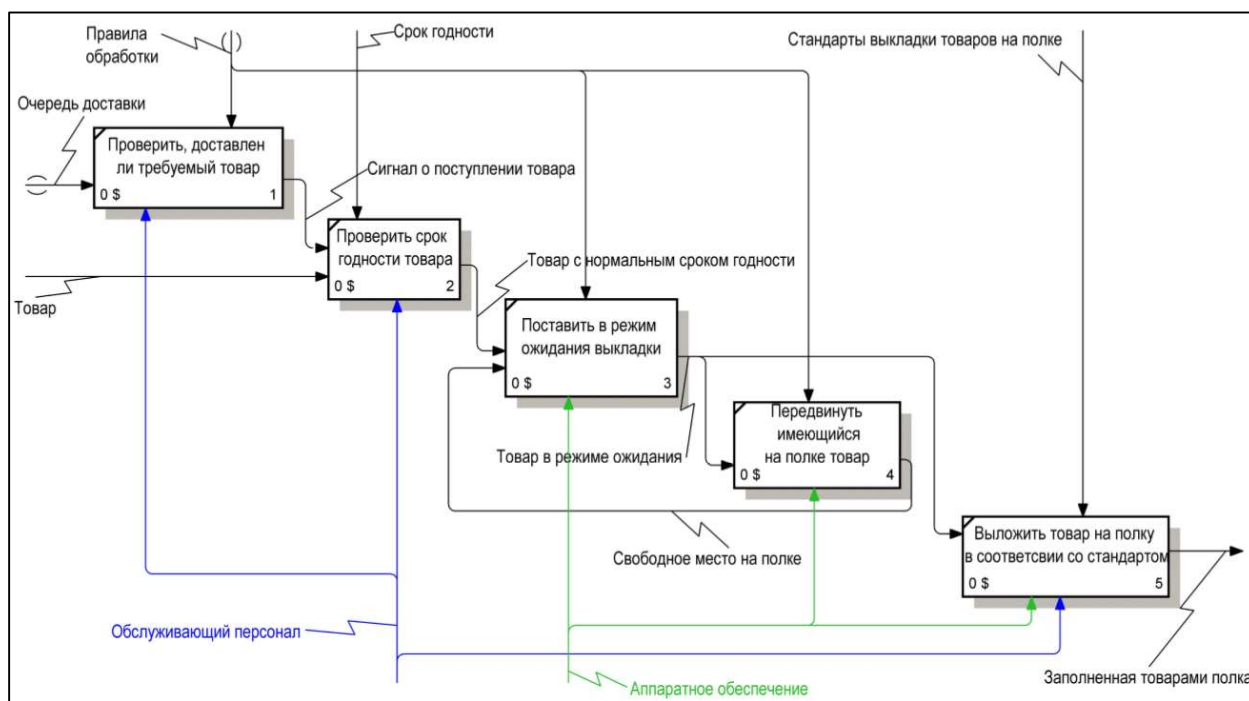


Рисунок 2.4 – Диаграмма декомпозиции для работы «Разместить поступивший товар на полке».

На вход поступает товар и информация об очереди доставки. Далее выдается сигнал о поступлении товара, товар определяется как товар с нормальным сроком годности, он переводится в режим ожидания, и когда появляется свободное место на полке, товар становится на нее. Все данные действия выполняются в соответствии с правилами обработки товара, информацией о его сроке годности и стандартами выкладки товаров на полке.

На выходе получается полка, заполненная товарами.

Все работы по анализу и обработке выполняются специально разработанным ПО, аппаратным обеспечением и, при некоторых условиях, мерчендайзерами, если у системы возникают проблемы или неполадки.

Для более полного описания системы были разработаны *DFD* - диаграмма и *IDEF3*-диаграмма для определенных блоков.

DFD – это нотация, предназначенная для моделирования информационный систем с точки зрения хранения, обработки и передачи данных. Для данной системы **DFD-диаграмма** была разработана для блока «Сгенерировать отчет со статистикой работы системы». Данная диаграмма представлена на рисунке 2.5.

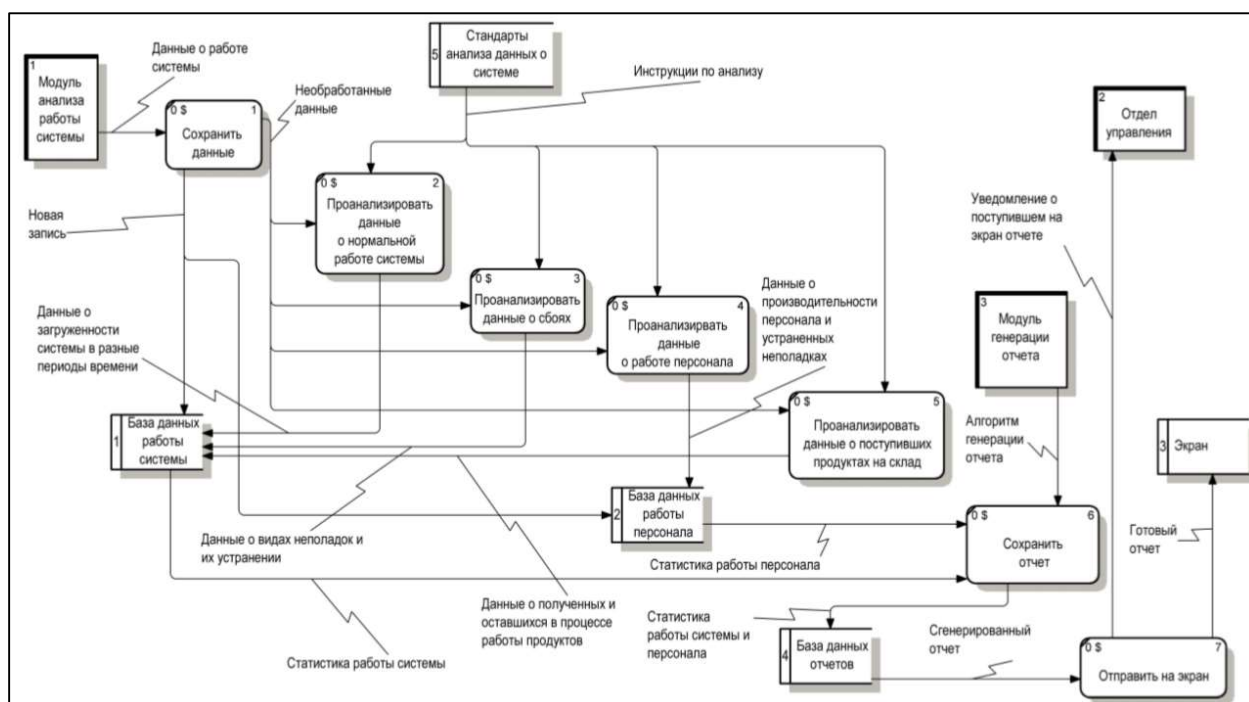


Рисунок 2.5 – DFD-диаграмма для блока «Сгенерировать отчет со статистикой работы системы»

На вход поступают данные, полученные в ходе работы системы. Они поступают на анализ в блоки анализа данных. Также создаются новые записи

в базе данных работы системы и базе данных работы персонала. Далее они анализируются в соответствии с инструкциями по анализу. После этого проанализированные данные сохраняются в этих же базах данных, откуда отправляются в виде статистик о работе системы и персонала. Далее они поступают в блок сохранения отчета и сохраняются в базу данных отчетов в виде объединенной статистики. Готовый отчет поступает в блок «Отправить на экран», выводится на него и отправляется в отдел управления.

Методология IDEF3 применяется для описания взаимодействия процессов (работ), т.е. порядка их выполнения, а также логических связей между ними. Диаграммы, построенные на основе методологии IDEF3 как и DFD-диаграммы, обычно строятся в качестве дополнения к IDEF0-диаграммам для их детализации. Для разрабатываемой системы была разработана *IDEF3-диаграмма* для блока «Подача сигнала о недостаточном наличии товара на полке». Разработанная диаграмма изображена на рисунке 2.6.

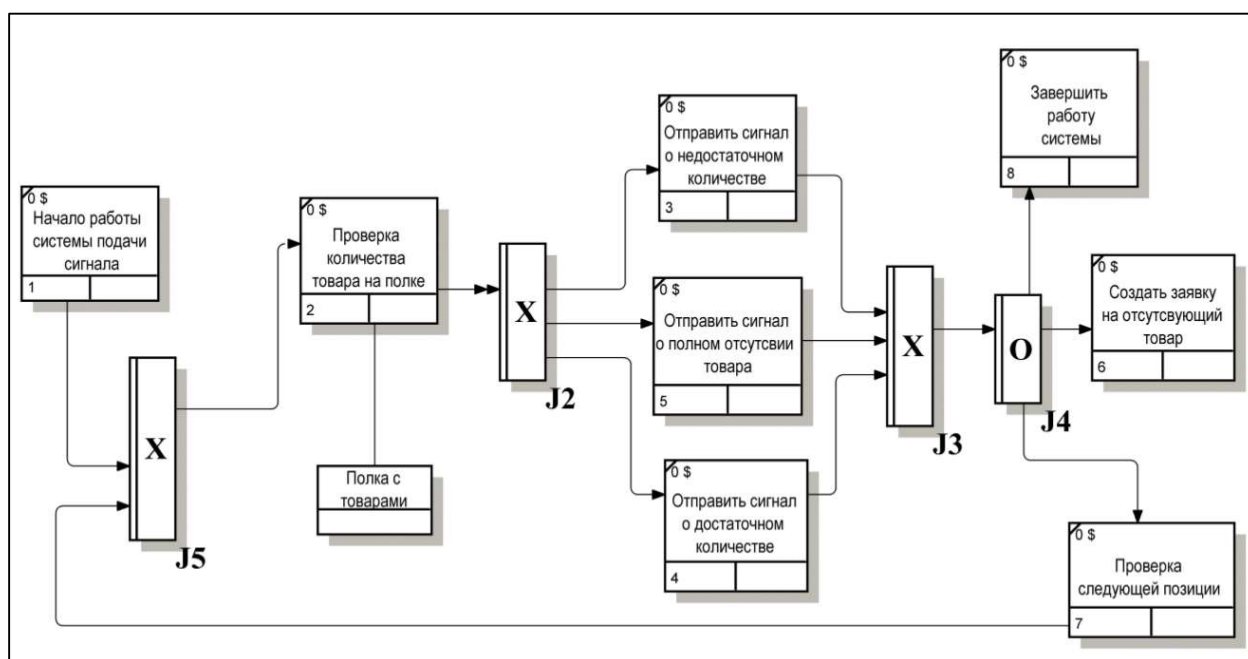


Рисунок 2.6 – IDEF3-диаграмма для блока «Подача сигнала о недостаточном наличии товара на полке».

Редактирование задачи начинается с проверки количества товара на полке. Для этого данные поступают с полки и анализируются. В зависимости от этого выполняется одна из работ: «Отправить сигнал о недостаточном количестве», «Отправить сигнал о полном отсутствии» или «Отправить сигнал

о достаточном количестве». После обработки одного из этих блоков, сигнал идет дальше на разветвление. Если система больше не нуждается в проверке и работе, сигнал идет на завершение работы системы. Если на полке недостаточное количество товара или он вовсе отсутствует, сигнал отправляется на создание заявки на соответствующий товар. Если товара достаточно, то сигнал отправляется на проверку следующей позиции. Все действия повторяются до завершения работы системы.

Для разработки диаграмм нотаций IDEF0, IDEF3, DFD использовалась среда проектирования ERwin Process Modeler.

### **2.1.3 Проектирование базы данных**

Базой данных (БД) называется организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность сведений об объектах, процессах, событиях или явлениях, относящихся к некоторой предметной области, теме или задаче. Она организована таким образом, чтобы обеспечить информационные потребности пользователей, а также удобное хранение этой совокупности данных, как в целом, так и любой ее части.

Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованы в виде набора таблиц, состоящих из столбцов и строк.

База данных предназначена для хранения данных о товарах, которые присутствуют или были в супермаркете, об их количестве на складе и на полках магазина, а также о дате их производства и сроке годности, указанном в днях с даты производства, о персонале супермаркета, а также о недавних грузоперевозках для анализа данных в соответствии с поступившими товарами и их остатке.

Пользователями системы, использующей БД, являются менеджеры супермаркета, осуществляющие управленческую деятельность и, в случае необходимости, предоставляют доступ к некоторым компонентам системы, если этого требует клиент, обслуживающий персонал, который состоит из нескольких мерчендайзеров, осуществляющих наблюдение за правильной расстановкой продуктов на полках, своевременной заменой продуктов с истекшим сроком годности и сообщающих о каких-либо неполадках в системе, системный администратор, которым может быть один из менеджеров супермаркета, а также клиенты, которые взаимодействуют с продуктами, расположенными на полках.

Функции системы, связанные с использованием БД: добавление новых товаров и данных о них, получение, добавление и редактирование данных о количестве товаров на полках и на складе, учет сотрудников и их должностей, а также управление их ролями, группами и правами в системе, хранение и управление данными о недавних грузоперевозках, хранение и управление данными об оборудовании, которое используется в процессе работы системы.

Требуется соблюсти соответствие базы данных трём нормальным формам. Нормальная форма – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц) [7].

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице. Например, для соблюдения 1НФ, создаётся таблица «Продукт», так как без неё в таблицах «Склад» и «Магазин» будет поле «Список продуктов», содержащее в себе более одного значения, остальные таблицы так же создаются с учётом требований 1НФ.

Исходя из функций и задач системы и требований 1НФ выделим сущности базы данных: Продукт (Product), Склад (Storage), Магазин (Store), Грузоперевозки (CargoTransportation), Персонал (Staff), Оборудование (Equipment). Так же при создании таблиц необходимо определить атрибуты сущностей (таблицы 2.2-2.7).

Таблица 2.2 – Сущность «Продукт»

Название	Описание
ID	Уникальный идентификатор
NAME	Название продукта
PRICE	Цена продукта
PRODUCTION_DATE	Дата производства продукта
EXPIRATION_DAYS	Количество дней срока годности продукта
PRODUCT_TYPE	Тип продукта

Таблица 2.3 – Сущность «Склад»

Название	Описание
ID	Уникальный идентификатор
PRODUCT_COUNT	Количество продукта на складе
DELIVERY_ID	Идентификатор грузоперевозки
PRODUCT_ID	Идентификатор продукта

Таблица 2.4 – Сущность «Магазин»

Название	Описание
ID	Уникальный идентификатор
SHELF_NUMBER	Номер полки магазина
PRODUCT_ID	Идентификатор продукта
PRODUCT_COUNT	Количество продукта на складе
MAINTAINER_ID	Идентификатор обслуживающего персонала

Таблица 2.5 – Сущность «Грузоперевозки»

Название	Описание
ID	Уникальный идентификатор
COMPANY_NAME	Имя компании, предоставляющей продукцию
DELIVERED_COUNT	Количество доставленного продукта
DELIVERY_DATE	Дата доставки продукта
PRODUCT_ID	Идентификатор продукта

Таблица 2.6 – Сущность «Персонал»

Название	Описание
ID	Уникальный идентификатор
NAME	Имя сотрудника
SURNAME	Фамилия сотрудника
POSITION	Должность сотрудника
GROUP_ID	Идентификатор принадлежности к группе пользователей

Таблица 2.7 – Сущность «Оборудование»

Название	Описание
ID	Уникальный идентификатор
NAME	Название оборудования
EQUIPMENT_TYPE	Тип оборудования
COST	Цена оборудования
LAST_MAINTAINANCE_DATE	Дата последнего обслуживания
WARRANTY_PERIOD_YEARS	Срок годности в годах
MAINTAINER_ID	Идентификатор обслуживающего персонала

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа (ПК). Для

соблюдения этой нормальной формы в таблицах «Склад» и «Магазин» создаются внешние ключи, а их сочетание составляет уникальный индекс. Таким образом, невозможна функциональная зависимость не ключевого атрибута от части первичного ключа.

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут не транзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы, в отдельные таблицы. Третья нормальная форма соблюдается за счёт создания таблиц «Продукт», «Персонал» и «Оборудование», чем мы исключаем возможность появления транзитивных связей.

Сущность «Склад» связана с сущностью «Продукт» связью «один ко многим», так как на складе может находиться несколько продуктов, но к продукту может относиться только одна позиция на складе. Кроме того, сущность «Склад» связана с сущностью «Грузоперевозки» для хранения идентификатора последней поставки продукта связью «один ко многим», так как один склад может хранить данные о нескольких последних грузоперевозках, но одна грузоперевозка может быть последней для одного продукта. Аналогично сущность «Магазин» связана с сущностью «Продукт» связью «один ко многим», так как к одной полке могут относиться несколько видов продуктов, но на одной позиции полки не может находиться больше одного продукта. Также сущность «Магазин» связана с сущностью «Персонал» связью «один ко многим», так как разные люди могут начать обслуживать одну полку, но одна полка одновременно принадлежит только одному человеку, обслуживающему ее.

Сущность «Грузоперевозки» связана с сущностью «Продукт» связью «один ко многим», так как перевозить можно разные виды продуктов, но за один раз перевозится только один вид продуктов. Сущность «Персонал» связана с сущностью «Группы» связью «один ко многим», так как один человек может одновременно принадлежать одной группе, но к группе может относиться сразу несколько человек. Сущность «Оборудование» связана с сущностью «Персонал» связью «один ко многим», так как один человек может взять на себя ответственность за несколько видов оборудования, но одновременно может обслуживать только один вид оборудования.

У каждой таблицы есть первичный ключ – поля уникальных идентификаторов.

В таблице «Склад» создаются внешние ключи: DELIVERY\_ID и PRODUCT\_ID, которые ссылаются на таблицы «Грузоперевозки» и «Продукт».

В таблице «Магазин» создаются внешние ключи: PRODUCT\_ID и MAINTAINER\_ID, которые ссылаются на таблицы «Продукт» и «Персонал».

В таблице «Грузоперевозки» создаётся внешний ключ PRODUCT\_ID, который ссылается на таблицу «Продукты».

В таблице «Персонал» создается внешний ключ GROUP\_ID, ссылающийся на таблицу «Группы».

В таблице «Оборудование» создается внешний ключ MAINTAINER\_ID, ссылающийся на таблицу «Персонал».

Моделирование данных представляет собой деятельность по обнаружению и документированию требований к информации. Требования к информации описывают данные и бизнес-правила, необходимые для поддержки бизнеса. Модель данных может выражать как сложные информационные потребности целой корпорации, так и конкретные информационные потребности одной единственной программы. Для наглядного представления базы данных была использована методология IDEF1х.

**Методология IDEF1X** подразделяется на уровни, соответствующие проектируемой модели данных системы. Каждый такой уровень соответствует определенной фазе проекта. Верхний уровень представляет собой логические модели и состоит из *Entity Relation Diagram* (Диаграмма сущность-связь) и *Key-Based model* (Модель данных, основанная на ключах). Нижний уровень состоит из *Transformation Model* (Трансформационная модель) и *Fully Attributed* (Полная атрибутивная модель).

Логическая модель данных является визуальным представлением структур данных, их атрибутов и бизнес-правил. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями. Проектирование логической модели должно быть свободно от требований платформы и языка реализации или способа дальнейшего использования данных. Логическая модель использует сущности, атрибуты и отношения для представления данных и бизнес-правил. Сущности представляют собой объекты, о которых корпорация заинтересована хранить данные. Атрибуты - это данные, которые корпорация заинтересована хранить. Отношения описывают взаимосвязи между сущностями в терминах бизнес-правил.



Для описания базы данных системы было решено разработать диаграмму верхнего уровня *Entity Relation Diagram*. Диаграмма сущность-связь является самым высоким уровнем в модели данных и определяет набор сущностей и атрибутов проектируемой системы. Целью этой диаграммы является формирование общего взгляда на систему для ее дальнейшей детализации. Ниже на рисунке 2.7 представлена ERD - диаграмма разрабатываемой базы данных.

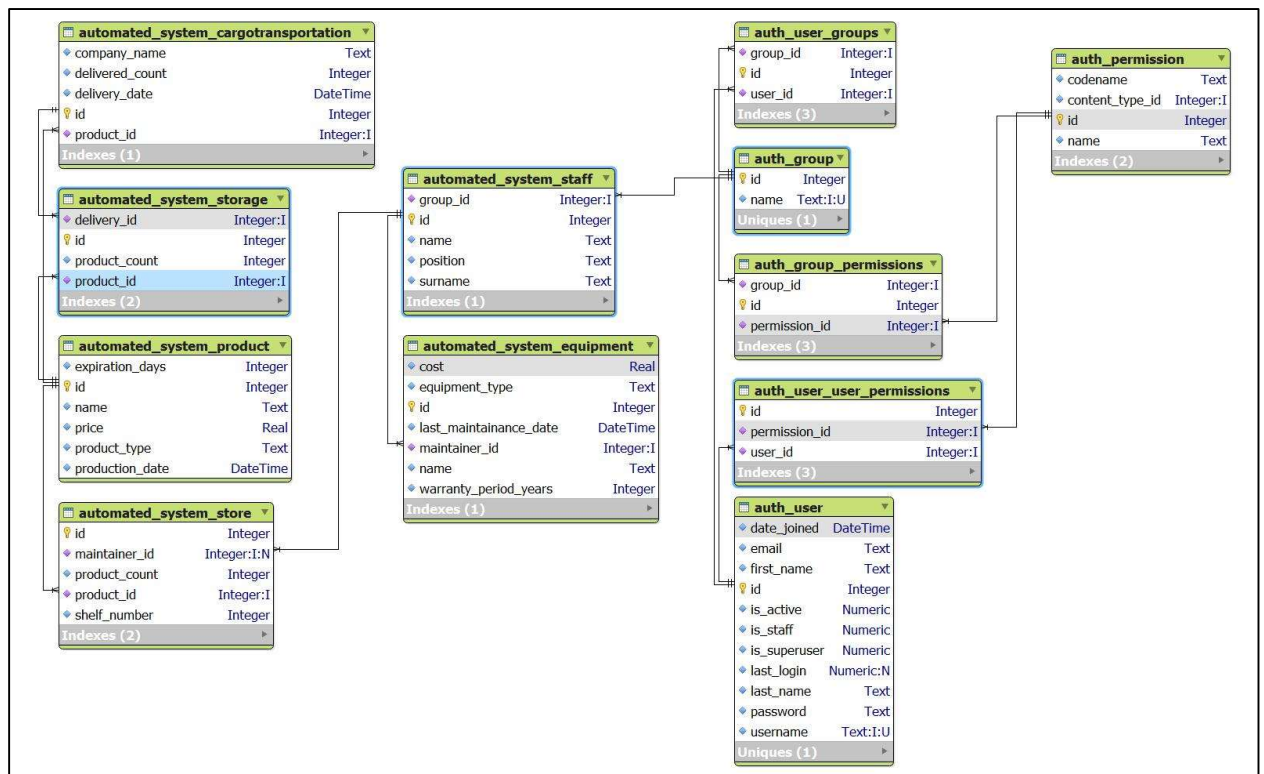


Рисунок 2.7 – ER-диаграмма базы данных

#### 2.1.4 Проектирование пользовательского интерфейса

Интерфейсы существуют, чтобы люди могли взаимодействовать с нашим миром. Через интерфейс мы можем прояснить, проиллюстрировать, дать возможность, показать взаимосвязь, объединить людей или разделить, управлять ожиданиями и давать доступ к услугам.

Дизайн пользовательского интерфейса является фактором, оказывающим влияние на три основных показателя качества программного продукта: его функциональность, эстетику и производительность.

Функциональность является фактором, на который разработчики приложений зачастую обращают основное внимание. Они пытаются создавать программы так, чтобы пользователи могли выполнять свои задачи и им было

удобно это делать. Функциональность важна, но, тем не менее, это не единственный показатель, который должен учитываться в ходе разработки.

Эстетичный внешний вид самого приложения и способа его представления (вплоть до упаковки) позволяет сформировать у потребителя положительное мнение о программе. Однако эстетические характеристики весьма субъективны и описать их количественно гораздо труднее, чем функциональные требования или показатели производительности. Вся эстетика приложения зачастую сводится к простому выбору: соотносятся ли между собой используемые цвета, передают ли элементы интерфейса их назначение и смысл представляемых операций, что ощущает человек при использовании тех или иных элементов управления и насколько успешно он их использует.

Производительность, а равно и надежность, также влияют на перспективу применения программы. Если приложение хорошо выглядит, имеет простое и удобное управление, но, к примеру, медленно прорисовывает экраны, регулярно «подвисает» на десяток-другой секунд или, того хуже, падает с критической ошибкой при некорректных действиях пользователя, у него, вероятно, будет мало шансов на длительную эксплуатацию. В свою очередь, быстрая и стабильная работа приложения могут отчасти компенсировать его не самый стильный дизайн или отсутствие каких-то вторичных функций.

К счастью, фреймворк Django предоставляет много разнообразных инструментов для создания красивого, эргономичного и быстродействующего интерфейса. Используя HTML и CSS можно построить уникальные страницы с удобными и плавными переходами, красивыми формами, разнообразными элементами управления и анимациями. Язык шаблонов Django позволяет создавать интерфейс практически не задумываясь о взаимодействии с серверным кодом, так как все необходимые элементы для передачи данных между этими двумя составляющими уже готовы, осталось их только использовать и встроить в код интерфейса пользователя.

## **3 РЕАЛИЗАЦИОННАЯ ЧАСТЬ**

### **3.1 Выбор инструментальной платформы и комплекса технических средств**

#### **3.1.1 Выбор СУБД**

Несмотря на то, что все системы управления базами данных выполняют одну и ту же основную задачу (т.е. дают возможность пользователям создавать, редактировать и получать доступ к информации, хранящейся в базах данных), сам процесс выполнения этой задачи варьируется в широких пределах. Кроме того, функции и возможности каждой СУБД могут существенно отличаться. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка.

При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также убедиться, что она будет хорошо интегрироваться с другими продуктами, которые уже используются. Кроме того, во время выбора следует принять во внимание стоимость системы и поддержки, предоставляемой разработчиком.

Сравниваются некоторые из популярных СУБД и на результате данного анализа выбирается подходящая для данного проекта.

##### **3.1.1.1 Microsoft SQL Server**

Это система управления базами данных, движок которой работает на облачных серверах, а также локальных серверах, причем можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска Microsoft SQL Server 2016, Microsoft адаптировала продукт для операционной системы Linux, а на Windows-платформе он работал изначально. Одной из уникальных особенностей версии 2016 года является temporal data support (временная поддержка данных), которая позволяет отслеживать изменения данных с течением времени. Последняя версия Microsoft SQL-сервер поддерживает dynamic data masking (динамическую маскировку данных), которая гарантирует, что только авторизованные пользователи будут видеть конфиденциальные данные.

Достоинства:

- продукт очень прост в использовании;
- текущая версия работает быстро и стабильно;
- движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
- вы сможете получить доступ к визуализации на мобильных устройствах;
- он очень хорошо взаимодействует с другими продуктами Microsoft.

Недостатки:

- цена для юридических лиц оказывается неприемлемой для большей части организаций;
- даже при тщательной настройке производительности корпорация SQL Server способен занять все доступные ресурсы;
- сообщается о проблемах с использованием службы интеграции для импорта файлов;
- идеально подходит для крупных организаций, которые уже используют ряд продуктов Microsoft.

### 3.1.1.2 PostgreSQL

PostgreSQL является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая Linux. Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

Движок БД может быть размещен в ряде сред, в том числе виртуальных, физических и облачных. Самая свежая версия, PostgreSQL 9.5, предлагает обработку больших объемов данных и увеличение числа одновременно работающих пользователей. Безопасность была улучшена благодаря поддержке DBMS\_SESSION.

Достоинства:

- является масштабируемым и способен обрабатывать терабайты данных;

- поддерживает формат json;
- существует множество predefined функций;
- доступен ряд интерфейсов;
- цена и открытость.

Недостатки:

- документация туманна, поэтому, возможно, ответы на некоторые вопросы придется искать в интернете;
- скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения;
- идеально подходит для организаций с ограниченным бюджетом, но квалифицированными специалистами, когда требуется возможность выбрать свой интерфейс и использовать json.

### 3.1.1.3 MySQL

MySQL - одна из самых популярных баз данных для веб-приложений. Фактически, является стандартом de facto для веб-серверов, которые работают под управлением операционной системы Linux. MySQL - это бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность. Существуют специальные платные версии, предназначенные для коммерческого использования. В бесплатной версии наибольший упор делается на скорость и надежность, а не на полноту функционала, который может стать и достоинством, и недостатком - в зависимости от области внедрения.

Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Эта СУБД позволяет выбирать различные движки для системы хранения, которые позволяют менять функционал инструмента и выполнять обработку данных, хранящихся в различных типах таблиц. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со

специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц. Она также имеет простой в использовании интерфейс, и пакетные команды, которые позволяют удобно обрабатывать огромные объемы данных. Система невероятно надежна и не стремится подчинить себе все доступные аппаратные ресурсы.

Достоинства:

- распространяется бесплатно;
- прекрасно документирована;
- предлагает много функций, даже в бесплатной версии;
- пакет MySQL включен в стандартные репозитории наиболее распространённых дистрибутивов операционной системы Linux, что позволяет устанавливать её элементарно;
- поддерживает набор пользовательских интерфейсов;
- может работать с другими базами данных, включая DB2 и Oracle.

Недостатки:

- придётся потратить много времени и усилий, чтобы заставить MySQL выполнять несложные задачи, хотя другие системы делают это автоматически, например, создавать инкрементные резервные копии;
- отсутствует встроенная поддержка XML или OLAP;
- для бесплатной версии доступна только платная поддержка;
- идеально подходит для: организаций, которым требуется надежный инструмент управления базами данных, но бесплатный.

### 3.1.1.4 SQLite

SQLite – это встраиваемая кроссплатформенная БД, которая поддерживает достаточно полный набор команд SQL и доступна в исходных кодах (на языке C). Исходные коды SQLite находятся в public domain, то есть вообще нет никаких ограничений на использование. Текущая последняя версия – 3.30.1.

Как известно, в своем развитии SQL устремился в разные стороны. Крупные производители начали добавлять свои расширения. И хотя принимаются стандарты (SQL 92), в реальной жизни все крупные БД не поддерживают стандартов полностью и имеют что-то свое. SQLite старается жить по принципу «минимальный, но полный набор». Она не поддерживает

сложные алгоритмы и продукты, но во многом соответствует SQL 92, и вводит некоторые свои особенности, которые очень удобны, но не стандартны для языка SQL.

Для SQLite не требуется сервер приложений. Доступ к БД происходит через «подключения» к БД (нечто вроде handle дескриптора файла ОС), которые мы открываем через вызов функции DLL. При открытии указывается имя файла базы данных, а если такого нету — он автоматически создается. Допустимо открывать множество подключений к одной и тоже базе данных (через имя файла) в одном или разных приложениях. Система использует механизмы блокировки доступа к файлу на уровне операционной системы, что может быть плохо, если работа с базой данных ведется через удаленный сервер. Изначально SQLite работал по принципу «многие читают — один пишет».

Достоинства:

- распространяется бесплатно;
- общедоступна и довольно распространена на многих платформах и операционных системах;
- ПО базы данных полностью покрыто автоматическими тестами;
- простая работа с использованием языка программирования;
- можно загрузить всю базу данных в память;
- легко переносить при смене системы или платформы;
- в любой столбец можно занести любое значение;
- множество встроенных функций.

Недостатки:

- не подходит для крупных производственных систем;
- нельзя удалить столбец в таблице;
- нет встроенной поддержки UNICODE;
- нет хранимых процедур;
- медленные операции манипуляции данными при большом количестве данных.

В целом, для решаемой задачи могут быть использованы MySQL, PostgreSQL или SQLite. Так как для MySQL потребуется устанавливать сервер базы данных, что несет дополнительную нагрузку на систему, а для PostgreSQL производительность может падать при частом считывании данных, и учитывая, что поддержка SQLite реализована на уровне фреймворка Django, выбор становится в пользу СУБД SQLite.

### 3.1.2 Выбор языка программирования

Существует множество языков программирования, предназначенных для выполнения различных задач. Каждый из них характеризуется уникальным набором операторов и особым синтаксисом.

При сравнении различных популярных языков программирования, следует учитывать стоимость системы и поддержки, предоставляемой разработчиком.

Сравниваются некоторые из популярных языков программирования и на результате данного анализа выбирается подходящая для данного проекта.

#### 3.1.2.1 JavaScript

JavaScript – это язык, который позволяет Вам применять сложные вещи на web странице — каждый раз, когда на web странице происходит что-то большее, чем просто её статичное отображение — отображение периодически обновляемого контента, или интерактивных карт, или анимация 2D/3D графики, или прокрутка видео в проигрывателе, и т.д. — можете быть уверены, что скорее всего, не обошлось без JavaScript.

JavaScript был создан в 1995 году в компании Netscape в качестве языка сценариев в браузере Netscape Navigator 2. Первоначально язык назывался LiveScript, но на волне популярности в тот момент другого языка Java LiveScript был переименован в JavaScript. Первоначально JavaScript обладал довольно небольшими возможностями. Его цель состояла лишь в том, чтобы добавить немного поведения на веб-страницу. Однако развитие веб-среды, появление HTML5 и технологии Node.js открыло перед JavaScript гораздо большие горизонты. Сейчас JavaScript продолжает использоваться для создания веб-сайтов, только теперь он предоставляет гораздо больше возможностей.

Также он применяется как язык серверной стороны. То есть если раньше JavaScript применялся только на веб-странице, а на стороне сервера нам надо было использовать такие технологии, как PHP, ASP.NET, Ruby, Java, то сейчас благодаря Node.js мы можем обрабатывать все запросы к серверу также с помощью JavaScript [8].

Преимущества JavaScript:

- ни один современный браузер не обходится без поддержки JavaScript;



- с использованием написанных на JavaScript плагинов и скриптов справится даже не специалист;
- полезные функциональные настройки;
- перспектива использования языка в процессе обучения программированию и информатике;
- постоянно совершенствующийся язык – сейчас разрабатывается бета-вариация проекта, JavaScript2.

Недостатки JavaScript:

- пониженный уровень безопасности ввиду повсеместного и свободного доступа к исходным кодам популярных скриптов;
- множество мелких раздражающих ошибок на каждом этапе работы. Большая часть из них легко исправляется, но их наличие позволяет считать этот язык менее профессиональным, сравнительно с другими;
- повсеместное распространение. Своеобразным недостатком можно считать тот факт, что часть активно используемых программ (особенно приложений) перестанут существовать при отсутствии языка, поскольку целиком базируются на нем.

### **3.1.2.2 Java**

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Первая версия языка появилась еще в 1996 году в недрах компании Sun Microsystems, впоследствии поглощенной компанией Oracle. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. И к настоящему времени язык Java проделал большой путь, было издано множество различных версий. На данный момент Java превратилась из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые в целого ряда задач. Кроме того, язык Java активно применяется для создания программного обеспечения для целого ряда устройств.

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы.

Java EE или Java Enterprise Edition представляет платформу для создания корпоративных приложений на языке Java. Прежде всего это сфера веб-приложений и веб-сервисов.

В 2017 году произошла новая веха в развитии платформы: Oracle передал контроль над развитием Java EE организации Eclipse Foundation. А в апреле 2018 года Java EE была переименована в Jakarta EE. [9]

Преимущества Java:

- гибкость. Java доказала, что С — процедурный, управляемый вручную и зависящий от платформы код — это не предел совершенства;
- переносимость. Программы, написанные на языке Java, после однократной трансляции в байт-код могут быть исполнены на любой платформе, для которой реализована виртуальная Java-машина;
- безопасность. Функционирование программы полностью определяется (и ограничивается) виртуальной Java-машиной. Отсутствуют указатели и другие механизмы для непосредственной работы с физической памятью и прочим аппаратным обеспечением компьютера;
- надежность. В языке Java отсутствуют механизмы, потенциально приводящие к ошибкам: арифметика указателей, неявное преобразование типов с потерей точности и т.п.;
- сборщик мусора. Освобождение памяти при работе программы осуществляется автоматически с помощью «сборщика мусора», поэтому программировать с использованием динамически распределяемой памяти проще и надежнее.

Недостатки Java:

- низкое, в сравнении с другими языками, быстродействие, повышенные требования к объему оперативной памяти;
- большой объем стандартных библиотек и технологий создает сложности в изучении языка;
- постоянное развитие языка вызывает наличие как устаревших, так и новых средств, имеющих одно и то же функциональное назначение.

### **3.1.2.3 Python**

Python широко применяется как интерпретируемый язык для скриптов различного назначения (хотя существуют и трансляторы языка Python).

Как и Ruby, Python имеет целью приблизить синтаксис реальной программы, написанной на нём, к описывающему задачу псевдокоду, что позволяет программисту уменьшить объём программы. Идея создания данного языка возникла в конце 1980-х и была реализована Гвидо ван Россумом.

Элегантный дизайн и эффективный, дисциплинирующий синтаксис этого языка облегчают программистам совместную работу над кодом. Python – мультипарадигмальный язык программирования: он позволяет совмещать процедурный подход к написанию кода с объектно-ориентированным и функциональным.

Преимущества Python:

- открытая разработка;
- довольно прост в изучении, особенно на начальном этапе;
- особенности синтаксиса стимулируют программиста писать хорошо читаемый код;
- предоставляет средства быстрого прототипирования и динамической семантики;
- имеет большое сообщество, позитивно настроенное по отношению к новичкам;
- множество полезных библиотек и расширений языка можно легко использовать в своих проектах благодаря предельно унифицированному механизму импорта и программным интерфейсам;
- механизмы модульности хорошо продуманы и могут быть легко использованы;
- абсолютно всё в Python является объектами в смысле ООП, но при этом объектный подход не навязывается программисту.

Недостатки Python:

- не слишком удачная поддержка многопоточности;
- на Python создано не так уж много качественных программных проектов по сравнению с другими универсальными языками программирования, например, с Java;
- отсутствие коммерческой поддержки средств разработки (хотя эта ситуация со временем меняется);
- изначальная ограниченность средств для работы с базами данных.

Если стоит вопрос о разработке web-приложения на Python, чаще всего человек приходит к выводу о том, что проще всего ее начать и поодерживать на фреймворке Django.

Django является чрезвычайно популярным и полнофункциональным серверным веб-фреймворком, написанным на Python. Когда у вас возникает определенная мысль, трансформировать ее на языке программирования и предать ей реальную форму при помощи Django займет всего несколько минут. То, что Django находится в свободном доступе, дает возможность

заметно упростить процесс веб разработки, так как разработчик может сфокусироваться на процессе дизайна и разработке функционала приложения. Кроме того, наличие встроенной поддержки SQLite дает возможность создавать качественные и чистые web-приложения очень быстро. Таким образом, Django – это идеальный инструмент для стартапов, когда веб дизайн должен отображать концепцию и цели компании [10].

Именно поэтому для демонстрации работы автоматизированной системы был выбран язык Python и база данных SQLite. Их интеграция находится на очень высоком уровне, что существенно облегчает разработку сайта системы.

В проекте задействован язык шаблонов, реализованный фреймворком Django. Он позволяет писать web-приложения с простой и понятной связью интерфейса пользователя и back-end кода. Язык шаблонов Django представляет баланс между возможностями и простотой. Он создавался, что бы быть удобным для пользователей HTML.

В качестве сервера приложений используется реализованный фреймворком Django внутренний сервер, который разворачивается вместе с запуском основного сервера Django. База данных представляет собой файл с расширением *sqlite3*, то есть нам не нужно задумываться о сервере базы данных, так как взаимодействие с ней уже налажено через фреймворк.

Django работает с десятками дополнительных функций, которые заметно помогают с аутентификацией пользователя, картами сайта, администрированием содержимого, RSS и многим другим. Данные аспекты помогают осуществить каждый этап веб разработки.

## **3.2 Программное обеспечение**

### **3.2.1 Структура программного обеспечения системы**

Основная идея архитектуры «клиент-сервер» состоит в разделении сетевого приложения на несколько компонентов, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом. Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

Автоматизированная система управления полочным пространством имеет веб-интерфейс, который позволяет взаимодействовать с ней через веб-страницы. Интерфейс построен при помощи языка гипертекстовой разметки HTML с применением технологии шаблонов Django и каскадных таблиц стилей CSS, а динамическая часть реализована на скриптовом языке JavaScript с использованием технологии AJAX.

Язык шаблонов Django позволяет пользовательскому интерфейсу напрямую взаимодействовать с back-end составляющей системы. Благодаря данной технологии не требуется задумываться о таких вещах, как написание адаптеров и дополнительных элементов чтобы связать логику системы и интерфейс. В связке с JavaScript и AJAX язык шаблонов позволяет создавать полноценные веб-приложения с динамическим обменом данными между интерфейсом пользователя и основным кодом программы. Кроме того, данная технология похожа на JSP, которая используется в Java/Jakarta EE, что позволяет быстрее ее освоить разработчику, ранее работавшему с JSP. На рисунке 3.1 изображена структура страниц проекта, которые доступны пользователю системы.

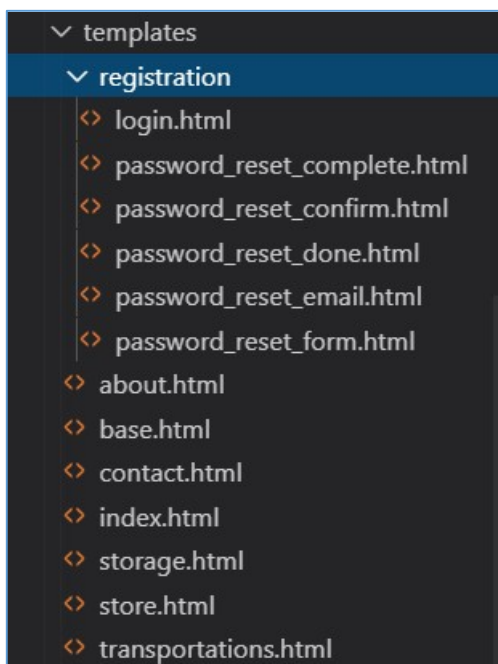


Рисунок 3.1 – Структура страниц проекта

### 3.2.2 Реализация ресурса

Для начала необходимо создать базу данных, чтобы прописывать логику приложения по работе с данными.

База данных создается с помощью ORM модели, которая реализована в Django. Таблицами базы данных являются модели, которые представляют из себя классы с описание свойств в виде переменных модели.

Ниже приведен код создания основных сущностей:

Таблица сущности «Продукт »:

```
CREATE TABLE "automated_system_product" (  
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "name" varchar(20) NOT NULL,  
    "price" real NOT NULL,  
    "production_date" datetime NOT NULL,  
    "expiration_days" integer NOT NULL,  
    "product_type" varchar(20) NOT NULL  
)
```

Таблица сущности «Грузоперевозки»:

```
CREATE TABLE "automated_system_cargotransportation" (  
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "company_name" varchar(100) NOT NULL,  
    "delivered_count" integer NOT NULL,  
    "delivery_date" datetime NOT NULL,  
    "product_id" integer NOT NULL REFERENCES  
    "automated_system_product" ("id") DEFERRABLE INITIALLY DEFERRED  
)
```

Таблица сущности «Оборудование»:

```
CREATE TABLE "automated_system_equipment" (  
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "name" varchar(20) NOT NULL,  
    "equipment_type" varchar(20) NOT NULL,  
    "cost" real NOT NULL,
```

```

        "last_maintenance_date" datetime NOT NULL,
        "warranty_period_years" integer NOT NULL,
        "maintainer_id" integer NOT NULL REFERENCES
"automated_system_staff" ("id") DEFERRABLE INITIALLY DEFERRED
    )

```

Таблица сущности «Персонал»:

```

CREATE TABLE "automated_system_staff" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "name" varchar(40) NOT NULL,
    "surname" varchar(40) NOT NULL,
    "position" varchar(50) NOT NULL,
    "group_id" integer NOT NULL REFERENCES "auth_group" ("id")
DEFERRABLE INITIALLY DEFERRED
)

```

Таблица сущности «Склад»:

```

CREATE TABLE "automated_system_storage" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "delivery_id" integer NOT NULL REFERENCES
"automated_system_cargotransportation" ("id") DEFERRABLE INITIALLY
DEFERRED,
    "product_id" integer NOT NULL REFERENCES
"automated_system_product" ("id") DEFERRABLE INITIALLY
DEFERRED,
    "product_count" integer NOT NULL
)

```

Таблица сущности «Магазин»:

```

CREATE TABLE "automated_system_store" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "shelf_number" integer NOT NULL,
    "product_id" integer NOT NULL REFERENCES
"automated_system_product" ("id") DEFERRABLE INITIALLY
DEFERRED,

```

```

        "product_count" integer NOT NULL,
        "maintainer_id" integer NULL REFERENCES "automated_system_staff"
        ("id") DEFERRABLE INITIALLY DEFERRED
    )

```

Далее идет реализация самой системы.

В начале создается файл base.html, который будет иметь общую структуру сайта: заголовок, фон, основные элементы CSS и обработку пользователя. С помощью языка шаблонов Django данная HTML-страница становится «шапкой» для всех остальных страниц. На ней загружаются статические файлы, которые будут использоваться на всех страницах:

```
{% load static %}
```

Далее идет фильтр, который проверяет, кто зашел на сайт:

```
{% load auth_filter %}
```

На данном этапе открытия страницы будет проверка пользователя, авторизован он или нет. Если пользователь не авторизован, ему выдается несколько информационных страниц, описывающих работу системы и предоставляющих контакты управляющего персонала. В случае, если пользователь авторизован, проверяется, к какой группе пользователей он принадлежит:

```

<ul class="navigation__list navigation__list--inline">
    <li class="navigation__item"><a href="/" {{
home_is_active|default:"" }}>Home</a></li>
    {% if user.is_authenticated %}
    <li class="navigation__item"><a href="/store" {{
store_is_active|default:"" }}>Store</a></li>
    <li class="navigation__item"><a href="/storage" {{
storage_is_active|default:"" }}>Storage</a></li>
    {% if request.user|has_group:'Management' or
request.user|has_group:'SystemAdministrators' %}
    <li class="navigation__item"><a href="/transportations"
    {{ transportations_is_active|default:""
    }}>Transportations</a></li>

```



```

        {% endif %}
    {% endif %}
    <li class="navigation__item"><a href="/about" {{
about_is_active|default:"" }}>About</a></li>
    <li class="navigation__item"><a href="/contact" {{
contact_is_active|default:"" }}>Contact</a></li>
    {% if user.is_authenticated %}
    <li class="navigation__item greeting">Hello, {{ user.get_username
}}</li>
    <li class="navigation__item"><a href="{% url 'logout'
%}">Logout</a></li>
    {% else %}
    <li class="navigation__item auth-button"><a href="{% url 'login'
%}">Login</a></li>
    {% endif %}
</ul>

```

Всего будет существовать три группы пользователей: администраторы, менеджеры и персонал. То есть на сайт могут зайти как авторизованные пользователи только работники супермаркета. Если пользователь является мерчендайзером, ему будут недоступны некоторые ссылки, а при попытке попасть на страницу, которую ему не разрешается видеть, будет выдаваться пустая страница с текстом, который говорит пользователю, что он не может видеть контент данной страницы.

Для встраивания другого HTML кода в данную страницу и использования ее в качестве шаблона для остальных страниц создаются блоки, в которых будет содержимое, характерное для других страниц:

```

{% block title %}Default title{% endblock title %}
{% block header %}{% endblock header %}
{% block content %}{% endblock content %}

```

Если на другой странице данного блока нет, он не будет меняться, то есть код на странице шаблона будет таким же, как и на этой странице.

Для подключения статических файлов на страницу используется тег `<link>`. Так как Django использует язык шаблонов, ссылка на статический файл указывается в виде инструкции: `{% static 'css/styles.css' %}`. Таким образом

указывается, что файл лежит в папке статического содержимого. Путь к папке указан в файле *settings.py*:

```
STATIC_URL = '/static/'
```

Файл *index.html* является главной страницей сайта. Попасть на нее можно при запуске сервера по ссылке: *http://127.0.0.1:8000/*. Данная страница расширяет функционал шаблонной страницы *base.html*:

```
{% extends "base.html" %}
{% block title %}Home{% endblock title %}
{% block content %}
<div id="welcome"><div>
{% endblock content %}
```

На данной странице пользователю выводится приветствие с просьбой авторизоваться для просмотра ее содержимого:

```
<h1>WELCOME TO AUTOMATED SHELF SPACE MANAGEMENT
SYSTEM</h1>
{% if user.is_authenticated %}
<div id="content"></div>
{% else %}
<p class="regular__item">You should be authenticated to continue work
with it</p>
{% endif %}
```

В случае, если пользователь авторизован, он сможет увидеть состояние системы. Если все в порядке, то пишется сообщение, помеченное зеленым цветом. В случае незначительных неполадок, не мешающих работе системы, выводится сообщение оранжевого цвета. Если нормальная работа системы невозможна – выводится сообщение красного цвета:

```
<p class="regular__item">System status: </p>
{% if current_status.store.status == "fine" %}
<p class="regular__item" style="color: green"><b>{{
current_status.store.data }}</b></p>
{% elif current_status.store.status == "warning" %}
```

```

    <p class="regular__item" style="color: orange"><b>{{
current_status.store.data }}</b></p>
    {% else %}
    <p class="regular__item" style="color: red"><b>{{
current_status.store.data }}</b></p>
    {% endif %}
    {% if current_status.storage.status == "fine" %}
    <p class="regular__item" style="color: green"><b>{{
current_status.storage.data }}</b></p>
    {% elif current_status.storage.status == "warning" %}
    <p class="regular__item" style="color: orange"><b>{{
current_status.storage.data }}</b></p>
    {% else %}
    <p class="regular__item" style="color: red"><b>{{
current_status.storage.data }}</b></p>
    {% endif %}
    {% if current_status.equipment.status == "fine" %}
    <p class="regular__item" style="color: green"><b>{{
current_status.equipment.data }}</b></p>
    {% elif current_status.equipment.status == "warning" %}
    <p class="regular__item" style="color: orange"><b>{{
current_status.equipment.data }}</b></p>
    {% else %}
    <p class="regular__item" style="color: red"><b>{{
current_status.equipment.data }}</b></p>
    {% endif %}
    {% if request.user|has_group:'Management' or
request.user|has_group:'SystemAdministrators' %}
    <p class="regular__item">Documents awaiting signature:</p>
    {% endif %}

```

Для авторизации используется страница *login.html*. Она также, как и все остальные страницы расширяет функционал шаблона *base.html*. На страницу авторизации также загружается статических файл стилей, который делает форму авторизации красивее, чем она была бы без него:

```

{% load static %}
<link rel="stylesheet" href="{% static 'css/auth_styles.css' %}" />

```

На данной странице пользователь может авторизоваться или восстановить пароль. Если пользователя не существует или он ошибся с вводом данных, страница его предупредит об этом:

```
{% if form.errors %}  
<p>Your username and password didn't match. Please try again.</p>  
{% endif %}
```

Если пользователь уже авторизован, страница не даст сделать этого повторно:

```
{% if user.is_authenticated %}  
<p>Your already logged in. </p>  
{% endif %}
```

Сама страница представляет из себя форму для ввода данных для авторизации. Для предотвращения CORS-атак предусмотрен *csrf*-токен, который запрещает межсайтовую подделку HTTP запроса:

```
<form method="post" action="{% url 'login' %}" id="login">  
  {% csrf_token %}  
  <fieldset id="inputs">  
    <label for="{{ form.username.id_for_label }}">Username:</label>  
    <div id="username">{{ form.username }}</div>  
    <label for="{{ form.password.id_for_label }}">Password:</label>  
    <div id="password">{{ form.password }}</div>  
  </fieldset>  
  <fieldset id="actions">  
    <input type="submit" id="submit" value="login">  
    <a href="{% url 'password_reset' %}">Lost password?</a>  
  </fieldset>  
</form>
```

Главная по своей сути страница системы — *store.html*. На ней отображается имитация системы, с которой можно взаимодействовать. Полки, представляющие собой элементы таблицы, содержат элементы, которые являются продуктами. Справа от полок есть кнопка «Взять продукт», которая удаляет крайний продукт с полки. Данные кнопки представляют собой

элемент формы, которая обрабатывается AJAX-запросом в JavaScript коде. Чтобы постоянно не обновлять страницу при изменении данных в базе, предусмотрен асинхронный запрос к back-end коду, сохраняющий количество товаров после взятия очередного с полки:

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script>
function sendAjaxForm(ajax_form) {
    productsCountInStore[ajax_form.slice(-1)]--
    let storeProducts = productsCountInStore[ajax_form.slice(-1)]
    let storageProducts = productsCountInStorage[ajax_form.slice(-1)]
    let id = ids[ajax_form.slice(-1)]
    console.log(id)
    $.ajax({
        url: '{% url "store" %}', //url страницы (action_ajax_form.php)
        type: "POST", //метод отправки
        data: { "product_id": id, "storageProducts": storageProducts,
"storeProducts": storeProducts },
        success: (result) => {
            console.log('success')
            return false
        }
    })
}
$(document).ready(function () {
    for (let i = 1; i < 9; i++) {
        $("#form" + i).submit(function (e) {
            e.preventDefault()
            sendAjaxForm('form' + i)
        })
    }
})
</script>
<form method='POST' id='form1'>
    {% csrf_token %}
    <button class='button7' onclick="take('shelf1')" id='take-
button1'>Take<br>= ■■■</button>
</form>
```

Когда товара на полке становится недостаточно, поступает запрос на восполнение полки, при этом статус товара на домашней странице отображается как «оранжевый», а кнопка взятия товара блокируется. Если продуктов недостаточно, оставшиеся выставляются на полку, а при отсутствии товаров статус становится «красным»:

```
function fillShelf(shelfId) {
  if(productsCountInStorage[shelfId.slice(-1)]>emptyCounts[shelfId.slice(-1)]) {
    productsCountInStore[shelfId.slice(-1)] = 10
    productsCountInStorage[shelfId.slice(-1)]-=emptyCounts[shelfId.slice(-1)]
    sendAjaxForm(shelfId)
    isReady[shelfId.slice(-1)] = false
    isFillRequest[shelfId.slice(-1)] = true
    fillIntervals[shelfId.slice(-1)] = setInterval(() => {
      let takeButton = document.getElementById('take-button' +
shelfId.slice(-1))
      takeButton.innerHTML = 'Filling<br>wait...<br>=■■■■'
      takeButton.disabled = true
      fill(shelfId)
    }, 1000)
  } else {
    sendNotificationNoProductsLeft(shelfId)
  }
}
```

Сами товары представляют собой элементы, оформленные через CSS. К ним добавляются статические картинки для красоты и наглядности:

```
<p class='regular__item'>Shelf №1<br>=====
=====</p>
<div class="lds-ellipsis" id='shelf1'>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
```

```

<div></div>
<div></div>
<div></div>
<div></div>
</div>
<div class="lds-ellipsis" id='shelf_up1'>
  <div></div><div></div><div></div><div></div><div></div>
  <div></div><div></div><div></div><div></div><div></div>
</div>
<p class='regular__item'>=====</p>

```

Страница *storage.html* предоставляет таблицу с опианием имеющихся продуктов на складе. Она выводит название продукта, его количество, последний привоз и его картинку:

```

<h1 class='regular__item'>Storage</h1>
<table style='width: 100%; border: 2px solid'>
  <thead>
    <tr>
      <td style='border: 2px solid;'><b>Product</b></td>
      <td style='border: 2px solid;'><b>Amount</b></td>
      <td style='border: 2px solid;'><b>Last delivery</b></td>
      <td style='border: 2px solid;'><b>Image</b></td>
    </tr>
  </thead>
  <tbody>
    {% for product in products_in_storage %}
      <tr>
        <td style='border: 2px solid;'>{{ product.product_name }}</td>
        <td style='border: 2px solid;'>{{ product.product_count }}</td>
        <td style='border: 2px solid;'>{{ product.last_delivery }}</td>
        {% with 'images/'|add:product.image_name|add:'.png' as image_static %}
          <td style='border: 2px solid;'></td>
          {% endwith %}
        </td>
      </tr>
    {% endfor %}
  </tbody>
</table>

```

```

</tbody>
</table>

```

Страница *transportations.html* доступна только пользователям, входящим в группы менеджеров или системных администраторов. Она предоставляет возможность просмотреть данные о грузоперевозках: какой продукт был привезен, какая компания занималась доставкой, дата грузоперевозки и количество привезенного продукта:

```

{% if request.user|has_group:'Management' or
    request.user|has_group:'SystemAdministrators' %}
<h1 class='regular__item'>Transportations</h1>
<table style='width: 100%; border: 2px solid'>
  <thead>
    <tr>
      <td style='border: 2px solid;'><b>Product</b></td>
      <td style='border: 2px solid;'><b>Company delivered</b></td>
      <td style='border: 2px solid;'><b>Delivery date</b></td>
      <td style='border: 2px solid;'><b>Delivery amount</b></td>
    </tr>
  </thead>
  <tbody>
    {% for transp in transportations %}
    <tr>
      <td style='border: 2px solid;'>{{ transp.product_name }}</td>
      <td style='border: 2px solid;'>{{ transp.company_name }}</td>
      <td style='border: 2px solid;'>{{ transp.delivery_date }}</td>
      <td style='border: 2px solid;'>{{ transp.delivered_count }}</td>
    </tr>
    {% endfor %}
  </tbody>
</table>
{% else %}
<h1 class='regular__item'>You are not authorized to see this page</h1>
{% endif %}

```



Также создаются страницы *about.html* с картинкой, описывающей работу всей системы, и *contact.html*, предоставляющей контакты управляющего персонала:

```
{% block title %}About{% endblock title %}
{% block content %}
<h1 class='regular__item'>About</h1>
<p class='regular__item'>This site is designed to facilitate the management
of an automated supermarket shelf space management system.</p>
<p class='regular__item'>System automates the process of filling the shelf
space, while reducing human participation, reducing its functions only to
monitoring the quality of the exhibited products.</p>
{% load static %}

{% endblock content %}
{% block title %}Contact{% endblock title %}
{% block content %}
<h1 class='regular__item'>Contact</h1>
<p class='regular__item'>For all questions write to email:<b>
JohnDoe@example.com</b></p>
<p class='regular__item'>Contact phone number:<b> 8-(800)-555-35-
35</b></p>
{% endblock content %}
```

Все взаимодействие визуальных элементов построено на вызове соответствующих функций-роутеров, находящихся в модуле *view.py*. Листинг кода модуля приведен в Приложении А.

Настройка отображения страниц по запросу пользователя через адресную строку обрабатывается паттернами, указанными в файле *urls.py*. Правило выбора действия основывается на точности указанного запроса, то есть если пользователь ввел запрос, соответствующий вызову двух действий, будет вызвано то, которое более конкретно определяется паттерном:

```
urlpatterns = [
    path("", views.index, name='home'),
    path('admin/', admin.site.urls),
    path('store', views.store, name='store'),
    path('storage', views.storage, name='storage'),
```

```

    path('transportations', views.transportations, name='transportations'),
    path('about', views.about, name='about'),
    path('contact', views.contact, name='contact'),
    path('accounts/', include('django.contrib.auth.urls'))
]

```

Взаимодействие кода с базой данных происходит с помощью ORM. Классы моделей для доступа к базе данных содержатся в модуле *models.py*. Способы взаимодействия базы данных и кода основаны на функциональной парадигме программирования и определяются в модуле *db\_management.py*.

Система реализована с использованием Django 3.0, Python 3.8.0, SQLite 3.30.1 и JavaScript.

### 3.3 Организационное обеспечение

Ниже приведен алгоритм действий пользователя при работе с готовым продуктом.

При загрузке сайта пользователь видит главную страницу системы, изображенную на рисунке 3.2.

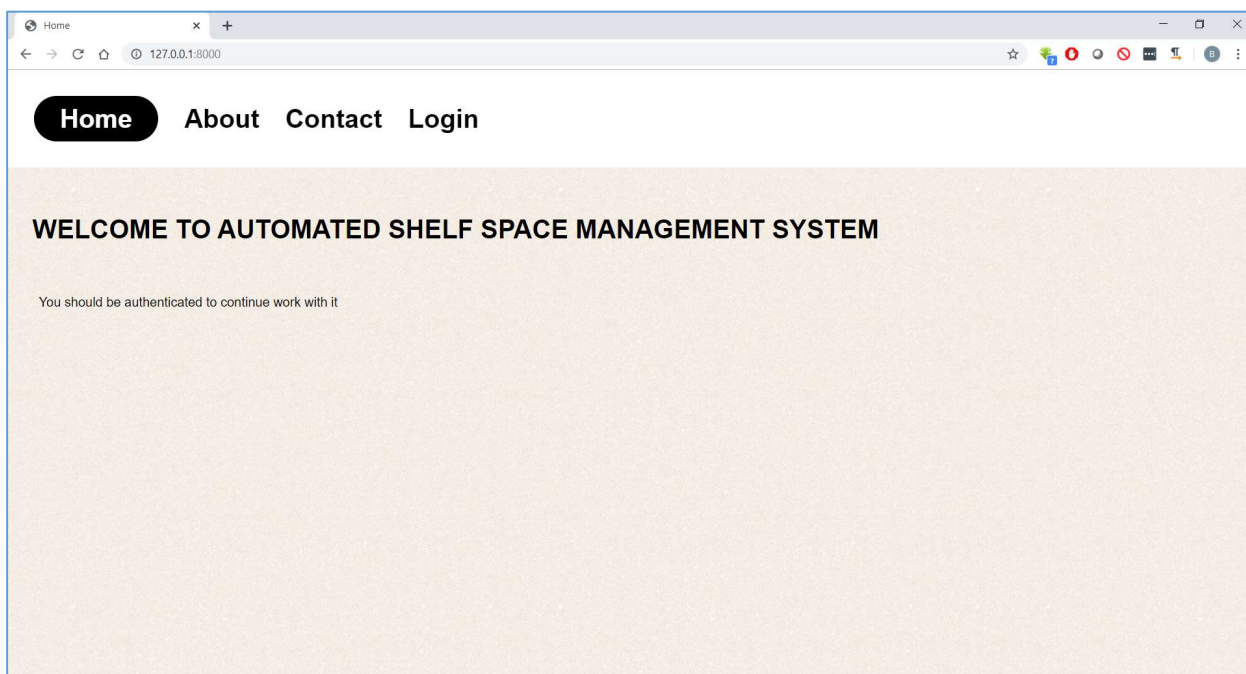


Рисунок 3.2 – Главная страница системы

Для дальнейшего взаимодействия с ней, необходима авторизация в целях защиты от несанкционированного доступа. Для этого пользователь может перейти на страницу авторизации, нажав на кнопку «Login». Страница авторизации представлена на рисунке 3.4.

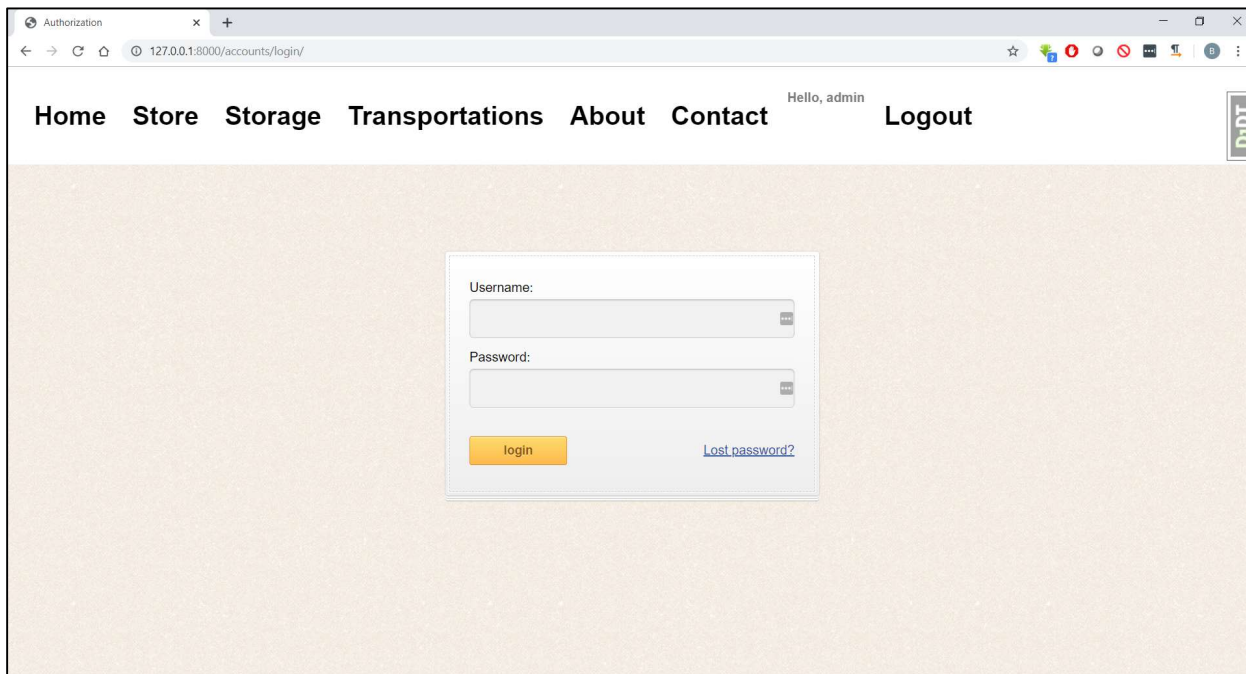


Рисунок 3.4 – Страница авторизации

После авторизации пользователь попадает обратно на главную страницу системы. С этого момента он будет иметь полный доступ к системе в соответствии с группой пользователей, к которой он принадлежит.

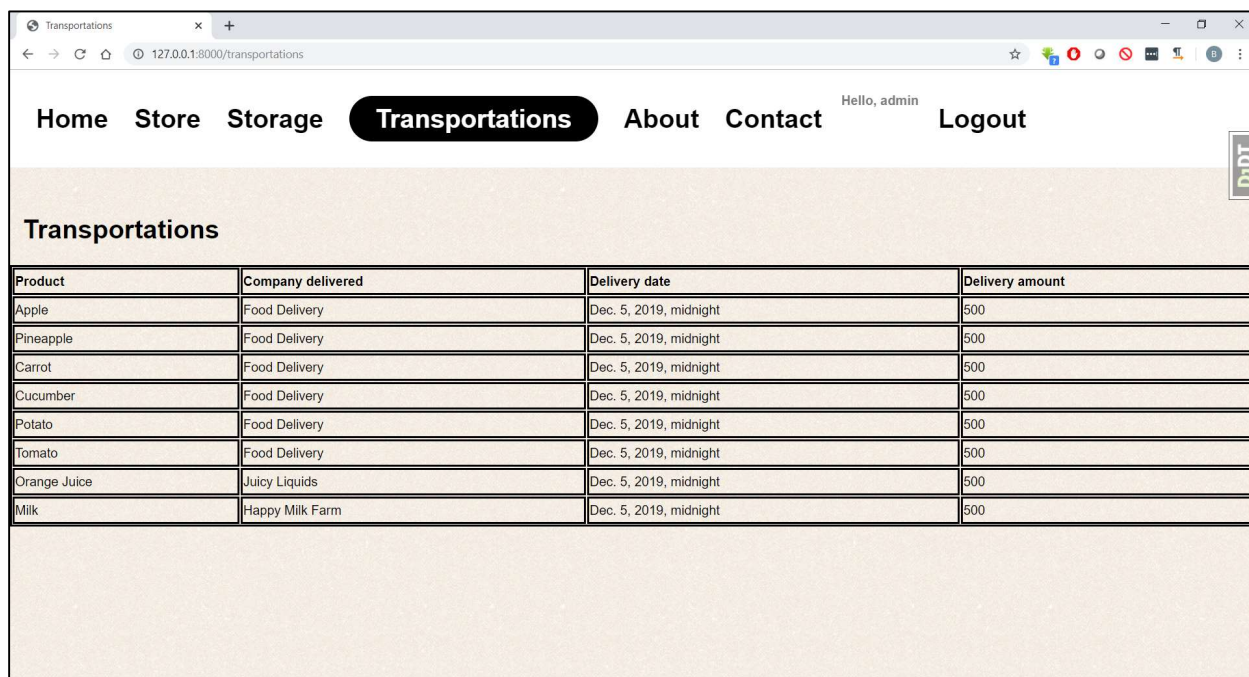
### 3.3.1 Руководство пользователя для менеджера и системного администратора

С помощью навигационной панели каждый пользователь может переходить на доступные ему страницы. Главное отличие менеджера и системного администратора от обычного пользователя – доступ к странице «*Transportations*» (рисунок 3.5).



Рисунок 3.5 – Навигационная панель менеджера

На данной странице можно увидеть список грузоперевозок и подробную информацию о том, какая компания предоставила продукт, когда была совершена грузоперевозка, какой товар был доставлен и в каком количестве (рисунок 3.6).



Product	Company delivered	Delivery date	Delivery amount
Apple	Food Delivery	Dec. 5, 2019, midnight	500
Pineapple	Food Delivery	Dec. 5, 2019, midnight	500
Carrot	Food Delivery	Dec. 5, 2019, midnight	500
Cucumber	Food Delivery	Dec. 5, 2019, midnight	500
Potato	Food Delivery	Dec. 5, 2019, midnight	500
Tomato	Food Delivery	Dec. 5, 2019, midnight	500
Orange Juice	Juicy Liquids	Dec. 5, 2019, midnight	500
Milk	Happy Milk Farm	Dec. 5, 2019, midnight	500

Рисунок 3.6 – Страница грузоперевозок

Кроме того, менеджеру доступна секция, отображающая документы, ожидающие подписи, на главной странице (рисунок 3.7).

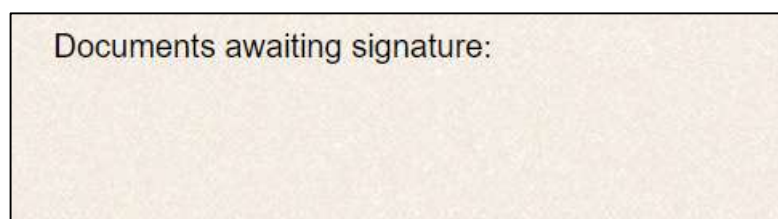


Рисунок 3.7 – Секция, отображающая документы, ожидающие подписи

### 3.3.2 Общее руководство пользователя

Все страницы, доступные рядовому сотруднику, представлены на рисунке 3.8.



Рисунок 3.8 – Навигационная панель сотрудника

После авторизации пользователь попадает на главную страницу, где ему выводятся сообщения о состоянии системы (рисунок 3.9).

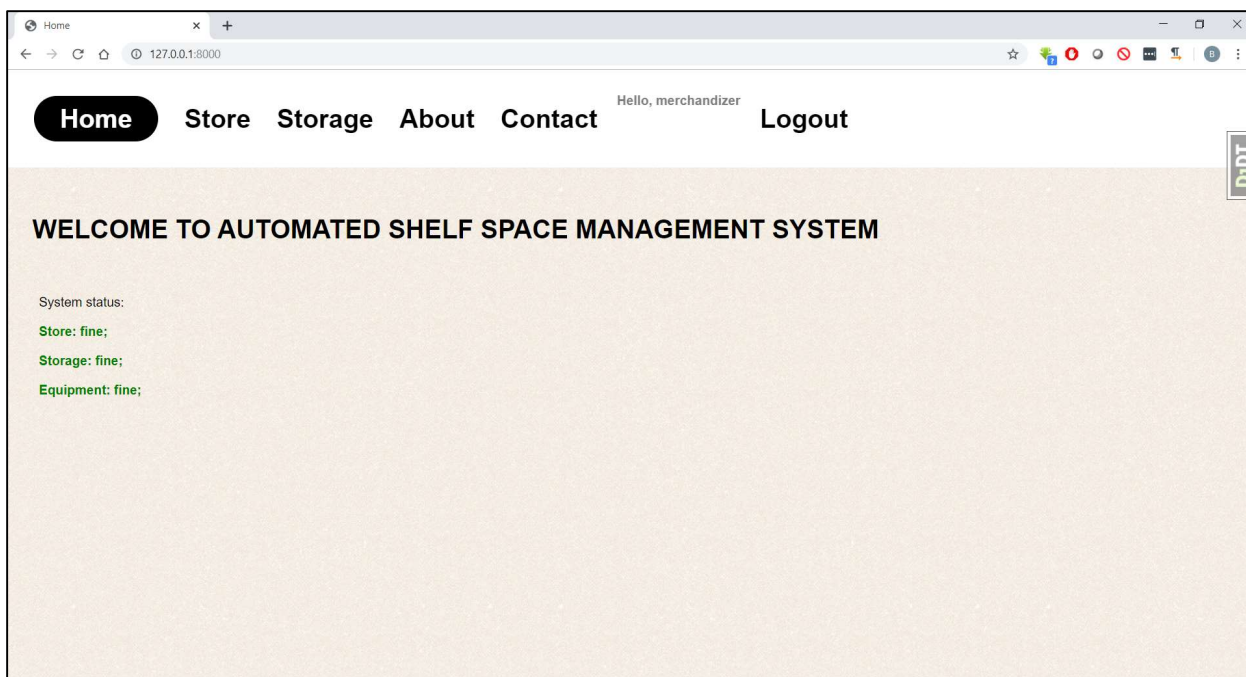


Рисунок 3.9 – Главная страница для авторизованного пользователя

Любой авторизованный пользователь имеет доступ к странице «*Store*», на которой отображается сама система и выполняются действия, имитирующие ее работу. На данной странице пользователю дается возможность взаимодействовать с системой (рисунок 3.10).



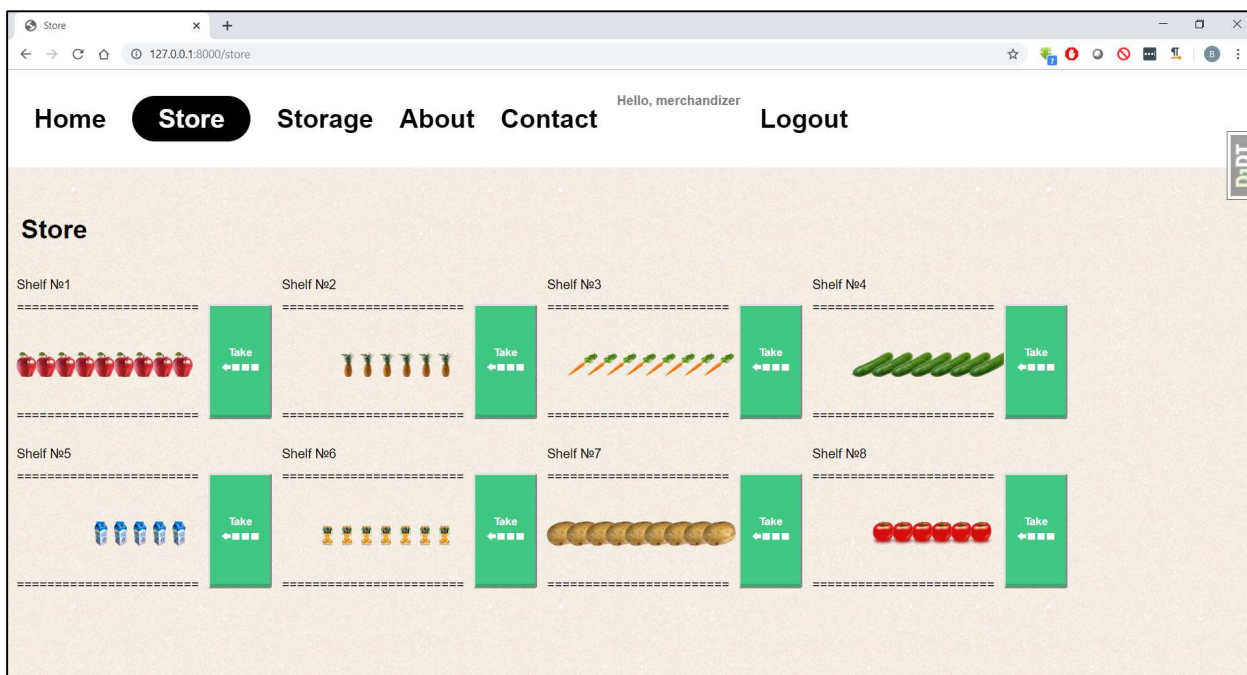


Рисунок 3.10 – Страница, имитирующая магазин

При начале пополнения полки, происходит взятие товара со склада и помещение его на полку. При этом кнопка временно блокируется до завершения данной операции (рисунок 3.11).



Рисунок 3.11 – Процесс заполнения полки

Изначально даже не авторизованный пользователь может посмотреть информацию о работе системы в виде очень краткого описания и картинки, иллюстрирующей это (рисунок 3.12).

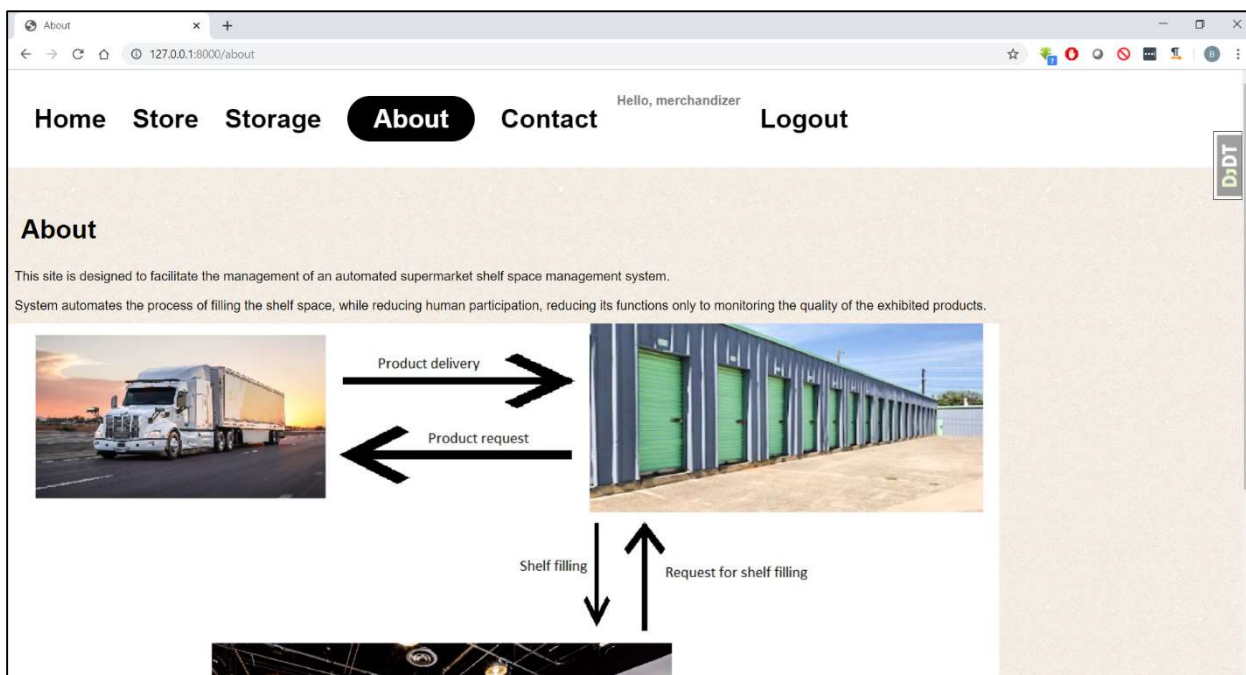


Рисунок 3.12 – Страница с информацией о работе системы

Также, любой пользователь может зайти на страницу с контактами менеджера, кликнув по кнопке «*Contact*» (рисунок 3.13).

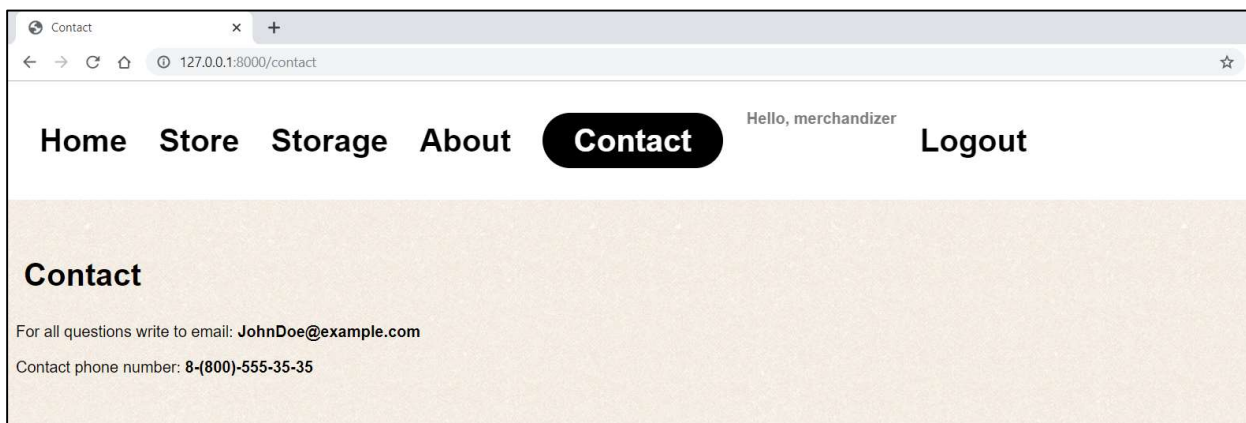
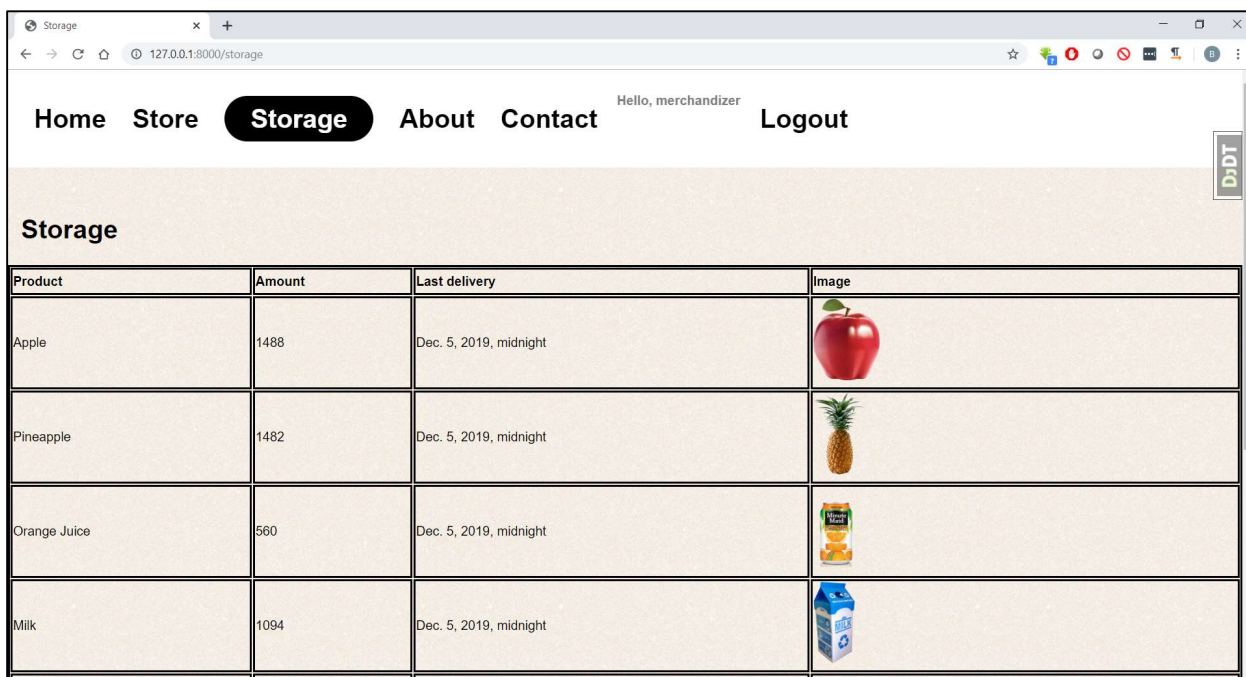


Рисунок 3.13 – Страница с контактами менеджера

Еще одной важной страницей, описывающей работу системы, является страница «*Storage*», которая отображает, какие продукты хранятся на складе, в каком количестве, когда они были завезены и как выглядят (рисунок 3.14).







Product	Amount	Last delivery	Image
Apple	1488	Dec. 5, 2019, midnight	
Pineapple	1482	Dec. 5, 2019, midnight	
Orange Juice	560	Dec. 5, 2019, midnight	
Milk	1094	Dec. 5, 2019, midnight	

Рисунок 3.14 – Страница с информацией о продуктах на складе

При нажатии на кнопку «*Logout*» в правом верхнем углу происходит завершение текущей сессии и возвращение на домашнюю страницу (рисунок 3.15).



Рисунок 3.15 – Кнопка «Logout»



## **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы была создана автоматизированная система управления полочным пространством супермаркета. Данный проект представляет собой пользовательский интерфейс, предназначенный для просмотра и взаимодействия с супермаркетом и его складом. Соблюдены все бизнес-требования к системе: разделение пользователей по правам доступа, возможность взять товар с полки магазина, автоматическое пополнение полки, извещение о неполадках, отображение информации о продуктах на складе и в магазине, просмотр информации о грузоперевозках. Бизнес-логика создана в соответствии с требованиями, представленными выше.

В дальнейшем планируется улучшить работу имитированного магазина, оптимизировать ее для большого количества товаров, протестировать в различных вариантах развития событий, добавить больше возможностей для управляющего персонала, создать генерацию отчета.

Соблюдены правила создания базы данных, в следствие чего предупреждены возможности неожиданного исчезновения или редактирования данных системой.

Так же соблюдались предписания по созданию пользовательского интерфейса, интерфейс получился интуитивно понятным и лояльным.

Использовались современные технологии по созданию веб-ресурсов, что обеспечило широкое отображение ресурсов, практически без ограничений по браузерной части.

Пояснительная записка оформлена в соответствии с требованиями стандарта СТП 01-2017.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Что такое автоматизация [Электронный ресурс] – Режим доступа: <http://elprivod.nmu.org.ua/ru/entrant/automation.php>, свободный.
- [2] Понятие автоматического и автоматизированного управления [Электронный ресурс] – Режим доступа: <https://lektsii.com/2-24538.html>, свободный.
- [3] Автоматизация розничной торговли, кассовое оборудование для розничной торговли [Электронный ресурс] – Режим доступа: <https://shtrih-m-spb.ru/articles/avtomatizatsiya-torgovli/>, свободный.
- [4] Построение системы управления полочным пространством - Доходность предприятия и пути ее повышения на примере ООО "Уралхимсервис" [Электронный ресурс] – Режим доступа: [https://studbooks.net/1993889/ekonomika/postroenie\\_sistemy\\_upravleniya\\_polochnym\\_prostranstvom](https://studbooks.net/1993889/ekonomika/postroenie_sistemy_upravleniya_polochnym_prostranstvom), свободный.
- [5] Система управления полочным пространством магазина, автоматизация управления выкладкой [Электронный ресурс] – Режим доступа: <https://abmcloud.com/abm-soft/shelf/>, свободный.
- [6] RS.ShelfSpace: система управления полочным пространством магазина, автоматизация мерчандайзинга, контроль выкладки товаров [Электронный ресурс] – Режим доступа: <https://shelfspace.ru/>, свободный.
- [7] Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование: учебное пособие. – БХВ-Петербург, 2009. – 528 с.
- [8] Введение в JavaScript [Электронный ресурс] – Режим доступа: <https://metanit.com/web/javascript/1.1.php>, свободный.
- [9] Введение в Java [Электронный ресурс] – Режим доступа: <https://metanit.com/java/tutorial/1.1.php>, свободный.
- [10] Плюсы и минусы Django [Электронный ресурс] – Режим доступа: <https://python-scripts.com/django-obzor>, свободный.
- [11] Вендров, А.М. Практикум по проектированию программного обеспечения экономических информационных систем : учебное пособие / А.М. Вендров. – Москва : Финансы и статистика, 2004. – 192 с. : ил. – (UML CASE) . – ISBN 5-279-02440-6.

## ПРИЛОЖЕНИЕ А

(справочное)

Листинг кода

**views.py**

```
from django.shortcuts import render
from django.http import HttpResponse, JsonResponse
from django.core import serializers
from .forms import UserForm
from .mgmt import db_manager
import json

is_active = "class=is-active"

def index(request):
    currentStatus = db_manager.get_current_status()
    userform = UserForm()
    return render(request, "index.html", {"form": userform,
"home_is_active": is_active, "current_status": currentStatus})

def store(request):
    if request.method == 'POST':
        print('custom post')
        db_manager.update_store_products(request.POST.get(
            'product_id'), request.POST.get('storeProducts'))
        print(request.POST.get('storeProducts'))
        print(request.POST.get('storageProducts'))
        db_manager.update_storage_products(request.POST.get(
            'product_id'), request.POST.get('storageProducts'))
        return HttpResponse("")
    else:
        storageProducts = db_manager.get_storage_products()
        storageProducts = serializers.serialize('json', storageProducts)
        storeProducts = db_manager.get_store_products()
        storeProducts = serializers.serialize('json', storeProducts)
        return render(request, "store.html", {"store_is_active": is_active,
"storageProducts": storageProducts, "storeProducts": storeProducts})
```

```
def storage(request):
    storageProducts = db_manager.get_storage_products_for_page()
    return render(request, "storage.html", {"storage_is_active": is_active,
"products_in_storage": storageProducts})
```

```
def transportations(request):
    transpos = db_manager.get_transportations_info()
    return render(request, "transportations.html",
{"transportations_is_active": is_active, "transportations": transpos})
```

```
def about(request):
    return render(request, "about.html", {"about_is_active": is_active})
```

```
def contact(request):
    return render(request, "contact.html", {"contact_is_active": is_active})
```

### **db\_manager.py**

```
from .. import models
from datetime import datetime
from datetime import date
```

```
def remove_product():
    models.Product.objects.filter(id=1).delete()
```

```
def update_store_products(product_id, product_count):
    models.Store.objects.filter(id=product_id).update(
        product_count=product_count)
```

```
def update_storage_products(product_id, product_count):
    models.Storage.objects.filter(id=product_id).update(
        product_count=product_count)
```

```
def update_references():
    models.Store.objects.filter(id=1).update(maintainer_id=3)
```

```
def checkStore():
    statusString = ""
```

```

status = "fine"
shelf = models.Store.objects.all()
for product in shelf:
    currentProduct = models.Product.objects.get(id=product.product_id)
    if product.product_count < 5:
        statusString = "Product: " + currentProduct.name + \
            ": insufficient count of product on shelf"
        if status != "error":
            status = "warning"
        print(product.product_count)
    elif product.product_count == 0:
        statusString = "Product: " + currentProduct.name + ": shelf is out
of this product"
        status = "error"
    if not statusString:
        statusString = "fine"
    return {"data": "Store: " + statusString + "; ", "status": status}

```

```

def checkStorage():
    statusString = ""
    status = "fine"
    storage = models.Storage.objects.all()
    for product in storage:
        currentProduct = models.Product.objects.get(id=product.product_id)
        if product.product_count < 50:
            statusString = "Product: " + currentProduct.name + \
                ": insufficient count of product in storage"
            if status != "error":
                status = "warning"
            print(product.product_count)
        elif product.product_count == 0:
            statusString = "Product: " + currentProduct.name + \
                ": storage is out of this product"
            status = "error"

    if not statusString:
        statusString = "fine"
    return {"data": "Storage: " + statusString + "; ", "status": status}

```

```

def checkEquipment():
    statusString = ""
    status = "fine"
    equipment = models.Equipment.objects.all()
    for item in equipment:
        currentDate = datetime.now()
        lastMaintainanceYear = item.last_maintainance_date.year
        warrantyExpirationDate = item.last_maintainance_date.replace(
            year=lastMaintainanceYear+item.warranty_period_years,
            tzinfo=None)
        daysToExpire = abs((currentDate-warrantyExpirationDate).days)
        if daysToExpire < 30 and daysToExpire > 0:
            statusString = "Equipment: " + item.name + \
                ": warranty is about to expire"
            if status != "error":
                status = "warning"
        elif daysToExpire < 0:
            statusString = "Equipment: " + item.name + \
                ": warranty expired"
            status = "error"
        if not statusString:
            statusString = "fine"
    return {"data": "Equipment: " + statusString + "; ", "status": status}

```

```

def get_current_status():
    storeStatus = checkStore()
    storageStatus = checkStorage()
    equipmentStatus = checkEquipment()
    return {"store": storeStatus, "storage": storageStatus, "equipment":
equipmentStatus}

```

```

def get_storage_products():
    return models.Storage.objects.all()

```

```

def get_storage_products_for_page():
    storageProduct = []
    for item in models.Storage.objects.all():

```

```

        product_name = models.Product.objects.get(id=item.product_id).name
        product_count = models.Storage.objects.get(id=item.id).product_count
        last_delivery = models.CargoTransportation.objects.get(
            id=item.delivery_id).delivery_date
        image_name = product_name.lower().replace(' ', '_')
        storageProduct.append({"product_name": product_name,
                                "product_count": product_count,
                                "last_delivery": last_delivery, "image_name":
image_name})
        return storageProduct

def get_store_products():
    return models.Store.objects.all()

def get_transportations_info():
    transpos = []
    for transp in models.CargoTransportation.objects.all():
        product_name =
models.Product.objects.get(id=transp.product_id).name
        transpos.append({"product_name": product_name, "company_name":
transp.company_name,
                        "delivery_date": transp.delivery_date, "delivered_count":
transp.delivered_count})
    return transpos

```

### **models.py**

```

from django.db import models
from django.contrib.auth.models import Group

class Product(models.Model):
    name = models.CharField(max_length=20)
    price = models.FloatField()
    production_date = models.DateTimeField()
    expiration_days = models.IntegerField()
    product_type = models.CharField(max_length=20, default="None")

class CargoTransportation(models.Model):
    company_name = models.CharField(max_length=100)

```

```
product = models.ForeignKey(Product, on_delete=models.CASCADE)
delivered_count = models.IntegerField()
delivery_date = models.DateTimeField()
```

```
class Storage(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    product_count = models.IntegerField(default=0)
    delivery = models.ForeignKey(CargoTransportation,
on_delete=models.CASCADE)
```

```
class Staff(models.Model):
    name = models.CharField(max_length=40)
    surname = models.CharField(max_length=40)
    position = models.CharField(max_length=50)
    group = models.ForeignKey(Group, on_delete=models.CASCADE,
default="")
```

```
class Store(models.Model):
    shelf_number = models.IntegerField()
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    product_count = models.IntegerField(default=0)
    maintainer = models.ForeignKey(
    Staff, on_delete=models.SET_NULL, null=True, default=None)
```

```
class Equipment(models.Model):
    name = models.CharField(max_length=20)
    equipment_type = models.CharField(max_length=20, default="None")
    cost = models.FloatField()
    last_maintenance_date = models.DateTimeField()
    warranty_period_years = models.IntegerField()
    maintainer = models.ForeignKey(Staff, on_delete=models.CASCADE,
default="")
```